Consulting & Analytics Club
IIT Guwahati

Gradient Ascent

2024-25

Question Bank

This guide covers questions and answers commonly asked in Data science and Machine learning interviews

Index

Basic Python

Q1. In Python, what is Polymorphism?

Answer: In Python, Polymorphism makes us understand how to perform a task in different ways in Python. It is useful in providing flexibility in task processes. Through polymorphism, a class's objects can invoke another class's methods, allowing for code reuse. Polymorphism also allows subclasses to override the methods of a superclass, allowing for further code reuse. This is especially useful in object-oriented programming, as it allows for inheritance that allows code to be written once and reused multiple times.

Q2. if Python is object-oriented or functional programming?

Answer: Python is considered to be a multi-paradigm language, which means it supports multiple programming techniques including object-oriented and functional programming. Since most Python tools have bundled up data and functions, it is considered to be object-oriented. The functions of Python are important for data scientists and programmers alike because Python supports both object-oriented and functional programming.

Q3. Python offers a powerful mechanism to extract specific parts of sequences like strings, lists, and tuples. This mechanism allows the programmer to specify a start, stop, and step to define the subsequence, without altering the original. While this technique works across various sequence types, its flexibility lies in its ability to handle both mutable and immutable sequences differently. What is this method called, and how can you use it to modify a sequence like a list?

Answer: This powerful mechanism in Python is known as slicing. It allows you to extract specific parts of sequences such as strings, lists, and tuples by specifying a start, stop, and step. It works well with all the sequence types and lets you manipulate your data cleanly and precisely without

changing the original sequence. Again, slicing can be applied to mutable sequences like lists. To change a section of a list, you can just assign new values to a slice:

```
my_list = [1, 2, 3, 4, 5]
my_list[1:3] = [7, 8] # Change elements starting from index 1 to index 2
print(my_list) # Output: [1, 7, 8, 4, 5]
```

So, slicing is pretty flexible to extract and alter data.

Q4. How does Python ensure that objects with the same name in different scopes don't conflict, and what structure does it use to maintain this system?

Answer: Python uses a system called namespace to provide unique identification for objects like variables and functions, ensuring no conflicts between names in different scopes. Namespaces are structured like dictionaries, where names are keys and objects are values. This system is maintained across local, global, and built-in scopes.

Q5. Which Python keyword allows you to leave code blocks such as loops or functions empty without causing a syntax error, and how does it differ from a comment?

Answer: The keyword pass allows you to leave some code blocks empty where code is syntactically required, such as in loops, conditionals, or even function definitions. Similar to comments, pass is recognized by the interpreter but it does nothing at runtime. This way, no syntax error of missing code statements will occur in your program.

Q6. What are ODBC modules in Python?

Answer: The Microsoft Open Database Connectivity is an interface for the C programming language. It is the standard for all APIs using database C. If you use a Python ODBC interface with the standard ODBC drivers that ship with most databases, you can likely connect your Python application with most databases in the market. The different Python ODBC modules are pyodbc, PythonWin ODBC, and MxODBC.

Q7.How will you send an email from a Python Script?

Answer: You can use a secure connection with the extensions SMTP_SSL() and .starttls(). Following this step, use the built-in smtplib library module to define the SMTP client session object. This

object can then be used to send the email message using Python Script. To send the emails you can use HTML content, as well as, the attachments with the email package. If you use a CSV file that contains contact data, you can even send a number of personalized emails. If you add a few lines of code to your Gmail account, you can configure the Yagmail package to send emails.

Q8. What is Self-used for in Python?

Answer: It is used to represent the instance of the class and allows you to access its attributes and methods. This is because in Python, the '@' syntax is not used to refer to the instance attributes.

Q9. What are generators in Python?

Answer: Generators are ways of implementing an effective representation of iterators and it is the only normal function that provides expression in the function. Thus, this enables Python developers to create iterators in a quick and clean way.

Q10. Differentiate between override and new modifiers.

Answer: The override modifier is used for overriding a base class function within the child class while the new modifier is used to inform the compiler to use the new implementation and not the base class function.

Q11. Why is NumPy preferred over Python lists?

Answer: The reason why NumPy is often chosen over Python lists is that Python lists are flexible, accept different data types, and therefore incur slow overhead, especially in massive-scale numerical operations. NumPy arrays, or ndarrays, are constructed on top of C arrays, statically typed, and homogeneous, which makes operations faster and uses memory efficiently.
One of the major advantages is the ability to perform vectorized operations. In other words, instead of using explicit loops as is common with the usage of Python lists, element-wise operations can be executed over whole arrays in one command. This is especially useful in the kind of tasks that you usually find in data science, like matrix operations in machine learning and statistical models.
Unlike Python lists, which are arrays of pointers, NumPy arrays are stored as blocks of contiguous memory. This has significantly reduced their memory usage and access time. Because of their homogeneous nature, compared to Python lists, NumPy arrays have lesser memory overhead since they store variable-sized and variable-typed elements.

In short, NumPy is a must-have due to its optimized storage, speed, and vectorized operations for efficient numerical computation in data science.

Q12. Python offers a way to transform objects into byte streams and later reconstruct them back into their original form. What are these two opposing processes called, and which functions handle them?

Answer: In Python, Pickling is the process of converting a Python object into a byte stream, which allows it to be saved to a file or transferred across networks. Actually, one of the most essential things for data persistence and transmission in real-world data applications. The pickle.dump() function is used for this purpose.

The reverse process, Unpickling, reconstructs the object from its serialized byte stream back into its native form again using pickle.load(). These techniques come out to be very helpful for working with machine learning models or very large datasets to have efficient storage and retrieval without having to recompute objects every time. This serialization process forms a significant part of model deployment and handling big data.

Q13. Explain database connection in Python Flask.

Answer: A SQLite3 command installation is needed to initiate and create the database in Flask. Using Flask, the database can be requested in three ways:

teardown_request() method: This is called in cases where the responses are not assured and the exception is raised.

after_request() method: This is called after requesting the database and also sending the response to the client.

before_request(): This method allows the database to be requested before only without passing arguments.

Q14. In high-traffic systems, when a cache expires and a surge of client requests overwhelms a website, what is this phenomenon called, and how can it be mitigated?

Answer: This is known as the Dogpile effect. It happens when a cache expires, and multiple clients simultaneously request the same data, overwhelming the system. To prevent this, a semaphore lock can be used. A semaphore is a synchronization mechanism that controls access to shared

resources in concurrent environments, such as threads or processes. It maintains a counter (typically an integer) that tracks the number of available resources. When a resource is requested, the counter decreases. If the counter reaches zero, subsequent requests must wait until the resource becomes available again.

In the case of cache rebuilding:

- When the cache expires, the first request triggers a semaphore lock, allowing only one process to regenerate the cache.
- Other requests that arrive during this time are blocked (or delayed) until the cache is rebuilt. Once the cache is refreshed, these requests are served using the new data, preventing multiple processes from hitting the backend simultaneously.

The semaphore works because it ensures only one process at a time performs the expensive operation of rebuilding the cache, while others wait, avoiding an overwhelming load on the system

Q15. Unlike many languages, Python doesn't explicitly use access specifiers. How does Python simulate the behavior of private and protected attributes, and what are the key symbols involved?

Answer: Python doesn't have traditional access specifiers like private, protected, or public. Instead, it uses naming conventions to mimic this behavior. A single underscore (_) before a variable or method name indicates it's intended for internal use (like protected), while a double underscore (__) triggers name mangling, simulating a private attribute by making it harder to access directly. Though these are just conventions, they help signal the intended scope of use within classes.

Q16. In Python, what do the notations *args and **kwargs signify, and how do they enhance the flexibility of function definitions?

Answer: Such notations as *args and **kwargs are powerful tools in handling variable numbers of arguments in Python functions.

- *args accepts an arbitrary number of non-keyworded arguments as a tuple. That is, you can pass any number of positional arguments without defining them explicitly in the function signature.
- **kwargs lets you input an arbitrary number of keyworded arguments in the form of a dictionary. This would allow you to pass named arguments that could then be processed in the function.

Together, they provide a great deal of flexibility because functions can accept many different types and numbers of input without strict definitions, which proves very useful in situations where the number of inputs is not predefined.

Python Libraries

Q1. import matplotlib.pyplot as plt
plt.plot([1, 2, 3], label="Line 1")
plt.show()

The plot appears, but the legend is missing. How can I fix it?

Answer: In Matplotlib, a simple plot with the labels does not prompt the legend to appear automatically. The plt.plot() function accepts a label argument, but by default, if you do not call plt.legend() after plotting, then the legend will also not be shown. So what is missing here is to add plt.legend(). This function creates the legend box in the plot. It is useful when you are plotting several lines or plots. This helps to make the visual more understandable by letting one distinguish one line or plot from another by their labels. Here is how you would do it: you simply add plt.legend() right after your plt.plot() call, then before your plt.show().

Q2. import matplotlib.pyplot as plt
img = plt.imread("sample_image.png")
plt.image(img)
plt.show()

Why is the image not displaying even though I used plt.image()?

Answer: The image does not show up because there is an incorrect function call for plt.image(). Rather than plt.image(), one should use plt.imshow() if they need to display an image in Matplotlib.The function imshow() is the short form for "image show." It has been defined only to portray images in Matplotlib. Indeed, there isn't any such function plt.image() so the image wasn't displayed. Now, after using plt.imshow(), the image is displayed properly on the figure.

Q3. I want to plot geographical data, but Matplotlib doesn't have a built-in function for maps. Which Python library should I use instead to plot maps?

Answer: Matplotlib isn't your first choice for geographical plotting since it isn't specialized for maps or geographical views. Generally, you would make use of either Geopandas or Basemap. Geopandas is an excellent extension of Pandas, and it's fantastic stuff about which you can do great things to create and plot geospatial data, including shapefiles and GeoJSON. Another great alternative is Folium, that works on top of Leaflet.js, the JavaScript library for interactive maps. Folium integrates nicely to Python and allows one to create interactive maps with little code. Alternatives for choropleth maps or where you need more control, there is also Plotly and Mapbox, including dynamic, web-based map creation. So, depending on your needs, use one of these over Matplotlib.

Q4.
```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2, 2)
axs[0, 0].plot([1, 2, 3])
```

Only one subplot appears instead of four. What is missing in this setup to correctly display multiple subplots?

Answer: Where we create multiple subplots using the function plt. subplots(), we actually create a grid of plots. The problem here looks like only the last subplot is populated and the rest are not utilized. The object axs returned by plt. subplots() is usually a 2D array, and you need to access and plot separately every subplot of this array. In this case, you'd want to ensure that each subplot within the grid gets filled with data. Thus, if you have a grid of 2 x 2, for instance, you would go about accessing each individual subplot using axs[0, 1], axs[1, 0], etc, and then plot something in each of them. Otherwise, just the first subplot will be filled; the rest will just be white.

Q5. What kind of statistical test is bartlett() used for, and how is it related to variance?

Answer: bartlett() test in Scipy is an analysis in statistics that tests the homogeneity of variances amongst groups. This is often simply referred to as Bartlett's Test for Equal Variances. The null hypothesis of this test of significance states that variances of the populations are equal. It is primarily useful when you have more than one set of data that you are comparing regarding their variances, as you want to know if they are significantly different or not. This has very important

implications for hypothesis testing in data science, because most statistical models assume homogeneity of variance, similar to ANOVA. If the variances are different, in general, this will negatively impact the performance of your model. Additionally, because Bartlett's test is sensitive to non-normality, if your data don't come from a normal distribution, you may want to use Levene's test instead.

Q6. _____ is a declarative library for data visualization.

1) Bokeh
2) Gleam
3) Altair
4) missingno

Answer: Altair uses a declarative style in creating plots, it becomes very easy and quick to repeat through visualizations and experiments at a fast pace when using this library. Here declarative means while plotting any chart, we declare connection between columns to the encoding channels, such as x-axis, y-axis, etc., and the rest related to plot details are handled automatically. we can do aggregation and filtering and dynamic filtering are the cool feature of python Altair. It gives a heat map that uses a text mark attribute. The code framework remains the same but changes in mark attribute can produce different plots. It supports line charts, stacked barplot.

Q7. What is the function name to draw the streamlines using Matplotlib package in Python?

Answer: The streamplot function is used to draw streamlines for vector fields, you will also have control of colour and width of the line. It helps the physicists to plot fluid flow and 2D field gradients.
Syntax: ax.streamplot(X, Y, u,v density=spacing)

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt

# Creating dataset
x = np.arange(0, 5)
y = np.arange(0, 5)
```

```
# Creating grids
X, Y = np.meshgrid(x, y)

# x-component to the right
u = np.ones((5,5))

# y-component zero
v = np.zeros((5, 5))

fig = plt.figure(figsize = (12, 7))

# Plotting stream plot
plt.streamplot(X, Y, u, v, density = 0.5)

# show plot
plt.show()
```

X and Y are 1D arrays on an evenly spaced grid,u and v are 2D arrays of velocities of x and y,density is a float value which controls the closeness of the stream lines.

Q8.
```
import matplotlib.pyplot as plt
plt.plot([1, 10, 100])
plt.yscale('log')
plt.show()
```

The plot is supposed to be logarithmic on the y-axis, but the plot appears linear. What could be going wrong?

Answer: In this case, the problem is probably coming from not setting the scale of the axis to logarithm. Matplotlib has provided two functions for setting the scales of the y-axis and x-axis respectively, plt.yscale('log') and plt.xscale('log'). However, when you have very small or negative values in your data, the log scale does not behave properly because the log of a negative number is undefined, and the log of zero is minus infinity. So you need to make sure that your data is strictly positive or possibly filter out or transform any zeros or negatives. That's typically where the problem is when working with log scales.

Q9. Given three different Seaborn plots - distplot(), boxplot(), and violinplot() - which of these plots are best suited for representing the distribution of categorical data? Besides, how do they treat data differently and in what cases one is preferable over the others?

Answer: All three are forms of categorical distribution plots: boxplot(), violinplot(), and boxenplot(). The most ubiquitous one is probably the boxplot(). A very easy way to get an idea of distribution by category for a continuous variable is to use the boxplot, which also gives useful statistics such as the median and quartiles, besides possible outliers. In fact, it's ideal to summarize and compare categories, getting a quick overview of two groups, for example.

violinplot() This is an extension of boxplot with a Kernel Density Estimate to give a fuller view of the distribution of data. It's handy when you think you might have multimodal data, or you want to see more than summary statistics. That is, for example, it's useful if a category has peaks in locations more than one, or if the distribution is left- or right-skewed.

boxenplot() is designed to handle large-sized sets of data more efficiently. It shows several percentiles in the data, which provides a finer detail than a single boxplot would do. This is especially useful when one works with huge sizes of data and wants to see a more fine grain distribution of values.

So Seaborn also has catplot() allowing you to combine these plots together with facets. This will make the kind of more complex plotting possible. You can specify the type of plot with e.g. box and it will automatically adapt the grid according to the variables.

So all three, the boxplot, violinplot, and boxenplot, are categorical distribution plots, and each has unique strengths depending on your type of data as well as the insights you are trying to draw.

Q10. import seaborn as sns

```
g = sns.FacetGrid(data, col="species")
g.map(plt.scatter, "sepal_length", "sepal_width")
```

The FacetGrid legend font size here is too small. How could you modify this to increase the font size?

Answer: First, I have to comment that FacetGrid does not automatically render the legend unless we request it to do so. Assuming there is a legend (for example, if we are using hue), we can adjust the font size by getting a reference to the legend object and updating its properties. I would do this:-

Now that we have the FacetGrid plot, we would need g.add_legend() in order to add a legend

within. Then, using the prop in add_legend() will allow me to directly manipulate and control the size of the fonts used for the legend.
g.add_legend() ensures the legend is not left out in the FacetGrid.
prop={'size': 12} allows me to set the text of my legend to a larger size by changing it to 12. You could alter this value to whatever size works for you.

Q11. Which popular Python library uses a grammar-of-graphics approach to simplify chart plotting, and how does this method differ from Seaborn or Matplotlib?

Answer: The grammar-of-graphics-based library is Plotly. The grammar-of-graphics approach was inspired by the ideas from Hadley Wickham's book for R, ggplot2, and allows for rendering graphics through layering different components in an organized manner. It helps in developing complex plots in a way that even an end user can have an intuitive approach to the methods applied in the process.
In the grammar-of-graphics framework, one begins with data and then defines aesthetic mappings, adds layers, and finally details the customization of plot components. This makes it easier for users to think systematically about the relationships between data elements, leading to a more systematic way to present data.
For instance, you can create the overall structure of the plot by making a high-level command with the type of data and aesthetic mappings. From there, you can add layers incrementally-like points, lines, and titles-add as though building up a sentence in a grammar system. This allows easily making adjustments-clear and logical-to advance your visualization without having to rewrite the entire code.
Seaborn and Matplotlib are more imperative in nature, but whereas it simplifies many tasks and enhances capabilities from Matplotlib, it still demands a more manual definition of what constitutes each component of your plot. For example, you have to specify everything explicit-from axes to the data points-in Matplotlib, which can result in pretty verbose syntax and sometimes not as intuitive for beginners.
For example, in Matplotlib, one would have to call several functions to create the figure, axes, and points; in Plotly, this can all be packaged into fewer, more intuitive commands. This can help you better be productive and read your code for even the most complex visualizations.
Another space where Plotly stands out is interactivity: plots produced with the library have an intrinsic, out-of-the-box kind of interactivity that can help stakeholders interact with data effectively. By contrast, even though Seaborn and Matplotlib can produce stunning static visualizations, they often require some additional effort or libraries' assistance to be interactive.
Putting all of this together, the grammar of graphics approach in Plotly is even more structured and flexible in creating visualizations which may be much more necessary for complex datasets. This

differs from more traditional approaches used in Seaborn and Matplotlib, where you have to manually handle each aspect of the plotting process.

Q12. import seaborn as sns
sns.bootstrap_plot([1, 2, 3])

Why is the function bootstrap_plot() not recognized in this code? How should you correctly create a bootstrap estimate?

Answer: The function bootstrap_plot() is unknown because it does not exist in the Seaborn library. Although Seaborn does offer various functions for any style of visualizations and statistical analyses, bootstrap_plot() is not in there. To run a bootstrap estimation in Python, you'll usually make use of the bootstrapped library or you can code your own using NumPy or SciPy.
Steps usually taken to make a bootstrap estimate are as follows:
Resampling: Take random samples with replacement from your data. This means you could pick a particular data point lots of times, or never pick a particular one.
Statistical Calculation: Calculate each resample using the statistic under consideration, such as the mean, median, or standard deviation.
Repeat: Repeat the resampling and statistical calculation lots of times-thousands-of times-to create distribution of the statistic.

For a basic example on how one can use NumPy, here's how to do a bootstrap estimate of the mean:

```
import numpy as np
data = [1, 2, 3]
n_iterations = 1000
bootstrap_means = []
for _ in range(n_iterations):
sample = np.random.choice(data, size=len(data), replace=True)
bootstrap_means.append(np.mean(sample))
# Calculate the mean and confidence intervals
bootstrap_mean = np.mean(bootstrap_means)
confidence_interval = np.percentile(bootstrap_means, [2.5, 97.5])
```

In this case, first we define our data as well as the number of bootstrap iterations. Then we sample from the original data with replacement, along with the calculation of the mean for each sample and storing it in a list. Finally, we can calculate the overall bootstrap mean and confidence intervals from the distribution of means.

Q13. import seaborn as sns
sns.factorplot(x="day", y="total_bill", data=tips)
Factorplot seems deprecated. What's the updated function I should use in Seaborn to create this categorical plot?

Answer: You are right; factorplot() is deprecated in the latest versions of Seaborn. Instead, use catplot().
The catplot() function is a flexible interface to making categorical plots and one may think of it more like a wrapper for various plot types that include bar plot, box plot, violin plot, and so on. Partly that's an evolution of Seaborn toward a more consistent and user-friendly API.

Here is how wecan do it:

```
import seaborn as sns
sns.catplot(x="day", y="total_bill", data=tips, kind="box")
```

Here, we are going to use catplot() in order to produce a box plot of total bills across different days from the tips dataset. The kind parameter lets you specify the type of plot you want, such as "box," "violin," or "bar."
This is also such a change that draws out such ease through usage of the same function for different types of categorical visualizations yet maintains clarity and consistency to your code.

Q14. How can I embed an interactive Bokeh plot within a Flask web app? Should I use a different strategy for Django? (+dev)

Answer: It's quite a simple task as Flask and Bokeh get along pretty well. Here's how I would do it:
Firstly I would generate the plot in Python using Bokeh's plotting functions.
Thereafter, embed the plot in Flask, the trick is using components() from Bokeh's bokeh.embed module. This will return the relevant HTML and JavaScript code to include the plot. Then you would render the plot using Jinja2 templating engine which comes with Flask.
Here is a brief example:

```
from flask import Flask, render_template
from bokeh.embed import components
from bokeh.plotting import figure
```

```
app = Flask(__name__)
@app.route('/')
def index():
plot = figure()
plot.circle([1, 2, 3, 4], [4, 7, 1, 6], size=20)
script, div = components(plot)
return render_template("index.html", script=script, div=div)
if __name__ == '__main__':
app.run(debug=True)
```

Rendering in HTML: In the HTML file (for example, index.html), you can add the plot with the script and div variables as follows:

```
<html>
<body>
{{ div | safe }}
{{ script | safe }}
</body>
</html>
```

This method ensures that the plot is rendered dynamically, and users can interact with it directly in the Flask app.
For Django: The overall strategy remains quite similar, but Django's templating system is a bit different. Instead of Flask's Jinja2, you would use Django's templating engine. The embedding process would still involve Bokeh's components() function.
But you'd normally put the Bokeh script and div into the template through context in the view function:

```
from django.shortcuts import render
from bokeh.embed import components
\\end
def index(request):
plot = figure()
plot.circle([1, 2, 3], [3, 2, 1], size=20)
script, div = components(plot)
return render(request, 'index.html', {'script': script, 'div': div})
```

Then in the Django template, you would then embed the Bokeh plot just like you have with Flask.

Key differences between the two
Flask:-

In this case, Flask uses functions for routing and rendering. Thus one creates a function which directly returns a template.
Django:-
In this case, Django relies on defining a view that will take the HTTP requests and return a response with the embedded Bokeh components.

Linear Regression

Q1. How can you check if the Regression model fits the data well?

Answer: We can use the following statistics to test the model's fitness.

- R-squared: It is a statistical measure of how close the data points are to the fitted regression line. Its value is always between a and 1. The closer to 1, the better the regression model fits the observations.
- F-test: It evaluates the null hypothesis that the data is described by an art intercept-only model, which is a regression with all the coefficients equal to zero versus the alternative hypothesis that at least one is not. If the P-value for the F-test is less than the significance level, we can reject the null hypothesis and conclude that the model provides a better fit than the intercept-only model
- Root Mean Square Error (RMSE): It measures the average deviation of the estimates from the observed value. How good this value is must be assessed for each project and context. For example, an RMSE of 1,080 for a house price prediction is probably good as houses tend to have prices over $100,000, but an RMSE of 1,000 for a life expectancy prediction is probably terrible as the average life expectancy is around 78

Q2. In what scenarios would one prefer Mean Absolute Error (MAE) over Mean Squared Error (MSE), and how do the mathematical properties of each metric influence their respective sensitivity to outliers?

Answer: The MAE and MSE both are measures of error, and while picking which one to use, in most cases, it depends on the specific goals of analysis and the characteristics of the dataset.

Scenarios for Preference

- Presence of Outliers: MAE is often preferred when the dataset contains significant outliers or when we expect the distribution to be multimodal. In such scenarios, MAE offers a better estimation of central tendency; therefore, the skewed values do not affect the measurement of the error metrics. Let's take an example that will be very useful to predict the prices of houses. In this scenario, with MSE, a couple of very high and low values may alter the performance of the model. With MAE, this will give a better view of what the average prediction accuracy might be.

MAE Formula:

$$MAE = \frac{1}{N}\sum_{i=1}^{n}|y_i - \hat{y}|$$

- Interpretability: MAE is intuitively better because it forms the average absolute difference between the predictions made by a model and the true values. It provides stakeholders with an intuition of the actual performance of the model in practice. As an example, if we say the MAE is $5,000, it directly notifies the reader that, on average, our predictions were off by this amount. This can easily be understood compared to an MSE number.
- MSE Formula:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

➔ Mathematical Properties:

- Sensitivity to outliers: MSE squares the residuals, so larger errors are penalized more severely. This feature is handy when outliers are to be important and we want our model to put strong priority on minimizing those larger deviations. For instance, in risk assessment models in which extreme

values can identify critical risks, MSE would ensure these risks were minimized.

- Linear vs. Quadratic Scaling: MAE is linear in the error whereas MSE is quadratic. This fundamental difference makes MSE a good choice for cases with normally distributed errors in scenarios in which we want to weight the larger errors more. However, if our dataset has errors that could be skewed, then MAE would better represent because it wouldn't exaggerate the influence of the outliers.

More in everyday language, my choice would depend on the context of the data involved, the sensitivity to outliers, and whether or not interpretability of results was relevant for the estimation.

Q3. What's the difference between Covariance and Correlation?

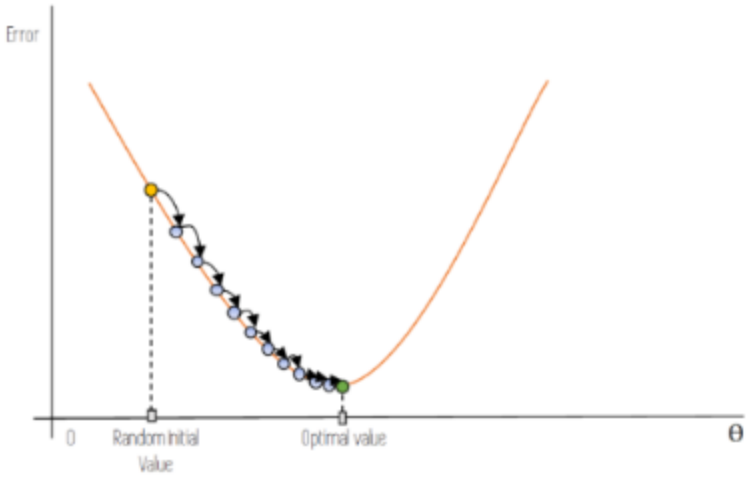
Answer: Covariance measures whether variation in one variable results in a variation in another variable, and deals with the linear relationship of only variables in the dataset. Its value can range from negative infinity to positive infinity Simply speaking Covariance indicates the direction of the linear relationship between variables.

- Correlation measures how strongly two or more variables are related to each other. Its values are between -1 to 1. Correlation measures both the strength and direction of the linear relationship between two variables. Correlation is a function of the covariance.

Q4. Explain the intuition behind Gradient Descent algorithm

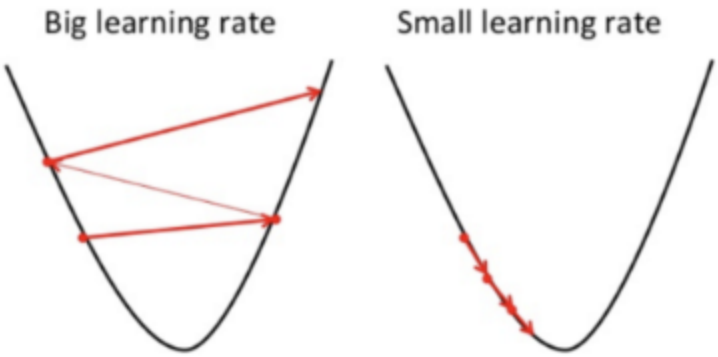
Answer: Gradient descent is an optimization algorithm that's used when training a machine learning model and is based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum (that is, slope = 0).

For a start, we have to select a random bias and weights, and then iterate over the slope function to get a slope of 0.



The way we change update the value of the bias and weights is through a variable called the learning rate. We have to be wise on the learning rate because choosing:

- A small leaning rate may lead to the model to take some time to learn
- A large learning rate will make the model converge as our pointer will shoot and we'll not be able to get to minima.



Q5. Explain the disadvantages of R-squared and how to address those?

Answer:

1. Overfitting: R-squared can be misleading when comparing models with different numbers of predictors. A higher R-squared value can be achieved simply by adding more independent variables, even if those variables are not significant. This can lead to overfitting, where the model performs well on the training data but poorly on unseen data.

   Addressing Overfitting:

- Use Adjusted R-squared, which adjusts the R-squared value for the number of predictors in the model. It accounts for the degrees of freedom and penalizes the addition of non-significant predictors. The formula for Adjusted R-squared is:

$$\text{Adjusted } R^2 = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - p - 1}\right)$$

   where n is the number of observations and ppp is the number of predictors.

2. Lack of Interpretability: R-squared does not provide insights into the individual contributions of each predictor or their significance. A high R-squared value does not imply that the model is appropriate or that the predictors are meaningful.

   Addressing Lack of Interpretability:

- Use statistical tests (like t-tests for coefficients) to evaluate the significance of each predictor. Additionally, employing other metrics such as p-values, confidence intervals, and feature importance can help provide a clearer picture of the relationships in the data.

3. Non-linearity: R-squared is not suitable for evaluating models that exhibit non-linear relationships between the dependent and independent variables. A high R-squared in such cases might lead to false confidence in the model.
   Addressing Non-linearity:
- Explore alternative modeling techniques like polynomial regression, decision trees, or non-linear models. Additionally, using residual plots to assess the distribution of residuals can help identify non-linear patterns that need to be addressed.

4. Sensitivity to Outliers: R-squared can be disproportionately influenced by outliers, which may distort the model fit and lead to misleading interpretations.

   Addressing Sensitivity to Outliers:

- Utilize robust regression techniques that are less affected by outliers, such as RANSAC (RANdom SAmple Consensus) or Huber regression. Checking residual plots for outliers and leveraging metrics like Mean Absolute Error (MAE) can also provide more resilient insights.

Q6. Provide an intuitive explanation of ransac regression algorithm.

Answer: RANSAC, or RANdom SAmple Consensus, is a robust regression technique known to be quite helpful when faced with data sets heavily contaminated by many outliers.

The algorithm starts by randomly selecting some small subset of the data. Ideally, this subset contains inliers—points fitting the model well—and then fits a model to this subset. RANSAC then assesses how many other points from the dataset are close to this model to make up a "consensus set" of inliers.

This process is repeated multiple times with different random samples, and at the end, the model which holds the largest consensus set is selected. This approach allows RANSAC to effectively ignore outliers and find a model best representing the trend in the data.

It is especially useful in applications involving computer vision, where noise and outliers are common. More generally, RANSAC provides a robust approach to model fitting that is least sensitive to anomalous data points.

Q7. How would you detect collinearity, and what is multicollinearity?

Answer: Multicollinearity is a kind of regression problem where two or more independent variables are said to be so correlated that they tend to provide redundant information about the response variable. The problem usually arises when there is instability in the coefficient estimates, because changes to very small effects in the model or data can have some significant influence on the outcome. Multiple methods can be used to detect multicollinearity:-

- Correlation Matrix: I would begin with the correlation matrix of the independent variables. High correlation coefficients (usually higher than 0.8 or 0.9) can indicate a possible occurrence of multicollinearity.
- Variance Inflation Factor (VIF): It is a more technical method. VIF determines to what extent a coefficient's variance is inflated by the existence of multicollinearity. VIF greater than 10 is often used as a rule of thumb for the presence of multicollinearity.

$$\text{VIF} = \frac{1}{1 - R^2}$$

Here, R^2 is the coefficient of determination of a regression of the variable in question on all the other independent variables.

- Condition Index: In this method, eigenvalues are calculated from the correlation matrix. A high condition index, usually greater than 30, implies multicollinearity problem.
- Scatter Plots: I would also plot scatter diagrams to visualize relationships between pairs of independent variables looking for evidence of a linear relationship between them that would indicate collinearity.

By using these techniques, I can check effectively for multicollinearity and take appropriate measures, such as deleting or collapsing variables, to make the model stronger.

Q8. Can you explain what an unrepresentative dataset is and how you would diagnose it?

Answer: An unrepresentative dataset is one that does not accurately reflect the population or phenomenon it is intended to model. This can occur due to several reasons, such as biased sampling, missing data, or over-representation of certain groups while under-representing others.

For example, if we were studying customer preferences in a city but only surveyed individuals from affluent neighborhoods, our dataset would likely be unrepresentative of the city's diverse population.

Diagnosing an Unrepresentative Dataset

1. Descriptive Statistics: I would start by analyzing basic descriptive statistics, such as means, medians, and standard deviations, across key variables. If these statistics differ significantly

from what is expected based on known population parameters, it might indicate unrepresentativeness.

2. Visualization: Creating visualizations like histograms, box plots, or bar charts can help me understand the distribution of data. Comparing these distributions to known population distributions can reveal discrepancies.
3. Sampling Method Analysis: I would examine the sampling method used to collect the data. If the sampling process was not random or if certain groups were systematically excluded, it could lead to an unrepresentative dataset.
4. Stratification: I might segment the dataset based on key demographics or characteristics and assess whether each segment is adequately represented. Under-representation of any segment can lead to biases in the overall analysis.
5. Comparative Analysis: If I have access to external datasets or benchmarks, I would compare the distribution of key features in my dataset to these sources. Significant differences could signal an unrepresentative sample.

By employing these diagnostic techniques, I can identify unrepresentative datasets and take corrective actions, such as collecting additional data or adjusting the analysis approach to ensure more accurate and reliable conclusions.

Q9. How would you detect heteroskedasticity in a regression model?

Answer: Heteroskedasticity refers to a condition in regression analysis where the variance of the errors is not constant across all levels of the independent variables. This can lead to inefficiencies in the estimates and affect hypothesis tests. Here are several methods I would use to detect heteroskedasticity:

1. Visual Inspection:
    a. Residuals vs. Fitted Plot: I would plot the residuals against the fitted values. If the spread of residuals increases or decreases with the fitted values (e.g., a fan or cone shape), this indicates heteroskedasticity.
2. Statistical Tests:
    a. Breusch-Pagan Test: This test checks for linear relationships between the squared residuals and the independent variables. A significant p-value indicates heteroskedasticity.
    b. White Test: This is a more general test that doesn't assume a specific functional form of the relationship between residuals and predictors. Like the Breusch-Pagan test, a significant result suggests heteroskedasticity.

c. Variance Inflation Factor (VIF): Although primarily used for multicollinearity, a high VIF can also suggest issues that may relate to heteroskedasticity, especially if predictors are correlated.
d. Goldfeld-Quandt Test: This test involves splitting the dataset into two groups and comparing the variances of the residuals. A significant difference suggests heteroskedasticity.

By utilizing these methods, I can effectively diagnose heteroskedasticity in a regression model, allowing for appropriate adjustments, such as transforming the dependent variable or using robust standard errors to obtain valid inference.

Q10. How would you address the problem of heteroskedasticity that arises from measurement error?

Answer: Heteroskedasticity due to measurement error can lead to biased and inefficient estimates in regression models. To address this issue, I would consider the following approaches:

1. Improved Measurement: The first step is to enhance the accuracy of data collection. Implementing more precise measurement tools or methods can reduce measurement error, leading to a more reliable dataset.
2. Instrumental Variables (IV): If the measurement error is systematic (i.e., affecting certain observations consistently), I might consider using instrumental variables. An instrumental variable is correlated with the independent variable but not directly related to the dependent variable. This can help provide consistent estimates in the presence of measurement error.
3. Error-in-Variables Models: I would consider using models specifically designed to handle measurement error, such as error-in-variables models. These models account for measurement error in the independent variables and can provide unbiased estimates of coefficients.
4. Weighted Least Squares (WLS): If the heteroskedasticity pattern can be identified, I might apply weighted least squares regression. By assigning weights to observations based on the estimated variance of the errors, WLS can stabilize the variance across the dataset.
5. Bootstrapping: If the measurement error is severe and difficult to quantify, I could use bootstrapping techniques to assess the variability of the estimates and obtain robust standard errors.

By employing these strategies, I can effectively mitigate the impact of heteroskedasticity caused by measurement error, ensuring that the regression model provides valid and reliable results.

Q11. How would you compare models using the Akaike Information Criterion (AIC)?

Answer: The Akaike Information Criterion (AIC) is a metric used to compare the goodness-of-fit of different models while penalizing model complexity. It's particularly useful in model selection when you're trying to balance fit and parsimony.

The AIC formula is:

$$\text{AIC} = 2k - 2\ln(L)$$

Where:

- k is the number of parameters in the model.
- L is the likelihood function, which measures how well the model fits the data.

To compare models using AIC:

1. Compute AIC: For each model, I would calculate the AIC value.
2. Lower AIC is better: The model with the lowest AIC value is considered the best because it strikes the best balance between fit (log-likelihood) and complexity (number of parameters).
3. Relative AIC Differences: While comparing, differences in AIC greater than 10 suggest strong evidence for preferring the model with the lower AIC. Differences between 0-2 suggest models are nearly indistinguishable, while 4-7 indicates some moderate evidence.

By using AIC, I can objectively select the best model without overfitting or unnecessarily complicating the model.

To compare models using AIC:

- Compute AIC: For each model, I would calculate the AIC value.
- Lower AIC is better: The model with the lowest AIC value is considered the best because it strikes the best balance between fit (log-likelihood) and complexity (number of parameters).

- Relative AIC Differences: While comparing, differences in AIC greater than 10 suggest strong evidence for preferring the model with the lower AIC. Differences between 0-2 suggest models are nearly indistinguishable, while 4-7 indicates some moderate evidence.

By using AIC, I can objectively select the best model without overfitting or unnecessarily complicating the model.

Q12. What are some advantages of using Gradient Descent over Ordinary Least Squares (OLS) for linear regression?

Answer: While OLS is a classic and efficient method for solving linear regression when the dataset is relatively small, Gradient Descent offers some key advantages in specific scenarios:

1. Scalability: Gradient Descent is more suitable for large datasets or high-dimensional data. OLS requires inverting a matrix, which can be computationally expensive as the dataset grows. Gradient Descent, on the other hand, works iteratively and is more memory-efficient.
2. Works with Non-Invertible Matrices: OLS can fail if the matrix of features (X) is singular or ill-conditioned. Gradient Descent doesn't require matrix inversion and can still provide a solution.
3. Flexibility with Optimization: Gradient Descent allows the use of various optimization techniques (e.g., Stochastic Gradient Descent, Mini-Batch Gradient Descent) and learning rates, making it adaptable to different problem settings and constraints.
4. Handles Regularization Easily: Gradient Descent can be easily extended to handle regularization techniques like Lasso (L1) or Ridge (L2), which add penalties to the model to avoid overfitting. This is more flexible than directly applying OLS in such cases.

Overall, Gradient Descent is preferred when dealing with large, complex datasets, or when regularization and scalability are key considerations.

Q13. How would you check if a linear model follows all regression assumptions?

Answer: To ensure a linear regression model is valid, I would verify it meets the following key assumptions:

1. Linearity: I would check if the relationship between the independent and dependent variables is linear. This can be done by plotting residuals vs. fitted values; any patterns in the plot suggest non-linearity.
2. Independence: To verify that residuals are independent, I might use the Durbin-Watson test, which checks for autocorrelation in the residuals. A value close to 2 indicates no autocorrelation.
3. Homoskedasticity: This assumes constant variance of residuals across all levels of the independent variables. I would check this using a residuals vs. fitted values plot. If the variance of residuals increases or decreases (like a fan shape), it indicates heteroskedasticity. I could also perform the Breusch-Pagan test.
4. Normality of Residuals: I would check whether residuals are normally distributed using a Q-Q plot or a histogram of residuals. Skewed or non-bell-shaped distributions may indicate a violation. The Shapiro-Wilk test can also be used for a formal test of normality.
5. No Multicollinearity: I would calculate the Variance Inflation Factor (VIF) for each independent variable. A VIF value above 10 indicates strong multicollinearity, which can distort model coefficients.

By systematically testing these assumptions, I can validate the appropriateness of my linear regression model.

Q14. How would you implement a linear regression function in SQL?

Answer: To implement linear regression in SQL, I would approach it by calculating the slope (β1) and intercept (β0) using the standard regression formulas. Here's how I'd go about it:-

- Calculate Means: First, I would compute the means of both the independent variable X and the dependent variable Y.
- Covariance and Variance: I would then calculate the covariance of X and Y, and the variance of X. This is key to determining the slope (β1).
- Formula for Slope and Intercept:
    - The slope β1 is calculated as the ratio of the covariance to the variance of X.
    - The intercept β0 is calculated by using the formula $\beta_0 = \text{mean}(Y) - \beta_1 \cdot \text{mean}(X)$
    - Here's the SQL query I'd use to implement it:

```
WITH summary_stats AS (
  SELECT
```

```
    AVG(X) AS avg_X,
    AVG(Y) AS avg_Y,
    SUM((X - AVG(X)) * (Y - AVG(Y))) OVER () AS covariance,
    SUM(POWER((X - AVG(X)), 2)) OVER () AS variance_X
  FROM
    your_table
)
SELECT
  avg_Y - (covariance / variance_X) * avg_X AS intercept, -- Calculate the intercept (β0)
  (covariance / variance_X) AS slope              -- Calculate the slope (β1)
FROM
  summary_stats;
```

This approach calculates the necessary summary statistics to derive the regression equation directly in SQL. It's scalable and avoids having to load the data into another tool. Once the slope and intercept are computed, we can use them to make predictions.

Q15. What are some challenges you might face when using a supervised regression model?

Answer: There are several challenges that can arise when using supervised regression models:

1. Overfitting: If the model is too complex, it may fit the training data well but fail to generalize to new, unseen data. Regularization techniques like Ridge or Lasso can help mitigate this by adding penalties to large coefficients.
2. Multicollinearity: When predictors are highly correlated, it can make the model unstable, causing inflated standard errors and unreliable coefficient estimates. Detecting multicollinearity using VIF (Variance Inflation Factor) and addressing it through dimensionality reduction methods like PCA or by removing correlated variables can help.
3. Heteroskedasticity: This occurs when the variance of errors is not constant across all levels of the independent variables, violating one of the key regression assumptions. It can lead to inefficient estimates. To address it, I'd use methods like weighted least squares or log-transforming the target variable.
4. Non-Linearity: Standard linear regression assumes a linear relationship between the predictors and the target variable. If the relationship is non-linear, the model will perform

poorly. Introducing polynomial features or using more complex models like decision trees or neural networks can help.

5. Outliers: Extreme values can disproportionately affect the model, especially when using error metrics like Mean Squared Error (MSE), which amplifies the impact of outliers. Robust regression techniques or metrics like Mean Absolute Error (MAE) can minimize their effect.
6. Data Quality and Missing Values: Poor data quality or missing values can hinder model performance. Imputation techniques or careful data preprocessing steps are necessary to address this.
7. Feature Selection: Including irrelevant features can lead to overfitting and increased model complexity. Techniques like backward elimination, regularization (Lasso), or feature importance from tree-based models can assist in selecting the most relevant features.

These are some common challenges, and handling them effectively often requires careful model evaluation and adjustments based on the data at hand.

Q16. Why would you use normalization versus standardization for linear regression?

Answer: The choice between normalization and standardization depends on the nature of the data and the model requirements.

1. Normalization: I'd use normalization (scaling features to a [0,1] range) when the data doesn't follow a Gaussian (normal) distribution or when I want to ensure all features contribute equally to the model. It's useful when we have variables of different units or scales and when outliers can disproportionately affect the model. For example, when using models like KNN or Neural Networks that depend on distance metrics, normalization helps to bring all features onto a similar scale.
2. Standardization: I'd apply standardization (z-score scaling, where the data is transformed to have a mean of 0 and a standard deviation of 1) when the model assumes a normal distribution of the features, as in the case of linear regression. Standardization is generally more robust to outliers, and because regression is sensitive to the magnitude of input features, standardization helps the algorithm converge faster during optimization.

In the context of linear regression, I'd typically lean towards standardization. This is because it makes gradient descent optimization more efficient, and it ensures that features with larger scales don't dominate the model's coefficients. Standardization is particularly helpful when dealing with models that involve regularization techniques like Ridge or Lasso.

Q17. How is hypothesis testing used in linear regression?

Answer: Hypothesis testing is fundamental to linear regression as it helps us evaluate the significance of relationships between independent and dependent variables.

First, we formulate two hypotheses: the null hypothesis (H_0), which states that there is no effect of an independent variable on the dependent variable (typically that the coefficient is zero), and the alternative hypothesis (H_a), which suggests that there is a significant effect (the coefficient is not zero).

Once we estimate the regression coefficients using methods like Ordinary Least Squares (OLS), we perform hypothesis tests for each variable. This involves calculating the t-statistic for each coefficient to assess how far the estimated value is from zero, expressed in terms of standard errors. The corresponding p-value helps us determine the likelihood of observing our results if the null hypothesis were true.

If the p-value is below a predetermined threshold, usually 0.05, we reject the null hypothesis, indicating that the variable has a statistically significant effect on the dependent variable. Conversely, if we do not reject the null hypothesis, it suggests that the variable may not significantly contribute to the model.

Additionally, we can assess the overall model fit using the F-test, which tests whether at least one predictor in the model is significantly associated with the outcome variable. This holistic approach, combined with confidence intervals for the coefficients, allows us to make informed decisions about the variables included in the model and their importance.

Overall, hypothesis testing in linear regression provides a rigorous framework for understanding the relationships in our data and guides our interpretation of the results.

Q18. Can you name some evaluation metrics for regression models and explain when you would use each?

Answer: There are several evaluation metrics that are commonly used to assess the performance of regression models, each suited for different contexts:

1. Mean Absolute Error (MAE): This metric calculates the average of the absolute differences between predicted and actual values. I'd use MAE when I want a clear interpretation of average errors in the same units as the target variable. It's robust to outliers, making it ideal when the data might contain extreme values.
2. Mean Squared Error (MSE): MSE averages the squared differences between predicted and actual values. It's particularly useful when I want to emphasize larger errors since squaring amplifies their impact. This is beneficial when my model needs to minimize significant errors.
3. Root Mean Squared Error (RMSE): RMSE is the square root of MSE, allowing me to interpret the error in the same units as the target variable. I would use RMSE when I want to understand the standard deviation of prediction errors, as it provides insight into the model's predictive accuracy.
4. R-squared (Coefficient of Determination): R-squared indicates the proportion of variance in the dependent variable explained by the independent variables. I find this metric helpful for assessing overall model fit, but I'm cautious since it can be misleading, particularly with multiple predictors.
5. Adjusted R-squared: This is a modified version of R-squared that accounts for the number of predictors in the model. I prefer using adjusted R-squared when comparing models with different numbers of predictors, as it provides a more accurate measure of performance by penalizing unnecessary variables.
6. Mean Absolute Percentage Error (MAPE): MAPE calculates the average absolute percentage error between predicted and actual values. I would use it when I want to express errors as a percentage, which can be more intuitive. However, I'd be careful with cases where actual values are zero, as it could become undefined.
7. Median Absolute Error (MedAE): MedAE gives the median of the absolute differences between predicted and actual values. I use it when I want a measure that is robust to outliers and provides a better summary in skewed datasets.

In summary, the choice of evaluation metric really depends on the specific context of the regression problem, the data characteristics, and how I want to interpret the results. Each metric provides unique insights that can guide model improvement and decision-making.

Logistic Regression

Q1. Why is Logistic Regression called regression and not Classification?

Answer: Although the task we are targeting in logistic regression is a classification, logistic regression does not actually individually classify things for you: it just gives you probabilities (or log odds ratios in the logit form).

The only way logistic regression can actually classify stuff is if you apply a rule to the probability output. For example, you may round probabilities greater than or equal to 50% to 1, and probabilities less than 50% to 0, and that's your classification.

Q2. What is the difference between linear and logistic regression?

Answer: Linear and logistic regression are both fundamental statistical techniques used in predictive modeling, but they serve different purposes and operate under different assumptions.

Nature of the Dependent Variable:

- Linear Regression: This is used when the dependent variable is continuous. For example, predicting house prices based on various features like size, location, and age. The output can take any real value, which means it can range from negative to positive infinity.
- Logistic Regression: This is used when the dependent variable is categorical, particularly binary. For instance, predicting whether an email is spam (1) or not spam (0). The output is a probability that can be transformed into a class label.

Model Equation:

- Linear Regression: The relationship between the independent variables and the dependent variable is modeled as a linear equation: $y=\beta0+\beta1x1+\beta2x2+\ldots+\beta nxn+\epsilon$ where y is the continuous output, β represents coefficients, x are the predictors, and ϵ is the error term.
- Logistic Regression: It uses the logistic function to model the probability that the dependent variable belongs to a particular category. The equation looks like this:

$P(Y=1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\ldots+\beta_n x_n)}}$, where P represents the predicted probability of the positive class.

Output Interpretation:

- Linear Regression: The output is interpreted as the expected value of the dependent variable for given independent variables. It provides a direct prediction.
- Logistic Regression: The output is interpreted as a probability, which can then be thresholded (usually at 0.5) to make a binary classification. For example, if the predicted probability is above 0.5, we classify it as the positive class.

Assumptions:

- Linear Regression: Assumes a linear relationship between independent and dependent variables, homoscedasticity (constant variance of errors), and normally distributed errors.
- Logistic Regression: Does not require a linear relationship between independent and dependent variables, but it assumes that the log-odds of the dependent variable is linearly related to the independent variables.

Error Measurement:

- Linear Regression: Typically evaluated using metrics like Mean Squared Error (MSE) or R-squared.
- Logistic Regression: Evaluated using metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.

In summary, while both linear and logistic regression are valuable tools in data analysis, they are applied to different types of problems based on the nature of the dependent variable and the relationships being modeled.

Q3. Compare Support Vector Machines (SVM) and Logistic Regression in terms of how they handle outliers.

Answer: Both Support Vector Machines (SVM) and Logistic Regression are popular algorithms for classification tasks, but they handle outliers quite differently.

1. Sensitivity to Outliers:

- Logistic Regression: This algorithm is generally sensitive to outliers because it tries to find a decision boundary that best fits the training data by minimizing the log loss. Outliers can significantly influence the coefficients, leading to a skewed model. For example, a single outlier can pull the decision boundary towards it, impacting the model's performance on normal data points.
- Support Vector Machines (SVM): SVMs are more robust to outliers, particularly in their traditional form. They focus on maximizing the margin between classes by considering only the support vectors (the data points that are closest to the decision boundary). Since outliers often do not fall within the margin, they have less influence on the model compared to Logistic Regression. However, SVM's performance can still be affected if outliers are present among the support vectors.

2. Decision Boundary:

- Logistic Regression: The decision boundary in Logistic Regression is determined by the coefficients learned from all data points. Since outliers can affect these coefficients, the resulting decision boundary may not generalize well to new, unseen data.
- Support Vector Machines (SVM): SVM constructs a decision boundary based on the support vectors and tries to create the widest margin possible. This means that if an outlier does not affect the support vectors, it won't significantly impact the model's performance.

3. Use of Regularization:

- Logistic Regression: Regularization techniques like L1 (Lasso) and L2 (Ridge) can help mitigate the impact of outliers by penalizing large coefficients. This can improve model robustness, but it requires careful tuning of the regularization parameter.
- Support Vector Machines (SVM): In SVM, the parameter CCC controls the trade-off between maximizing the margin and allowing misclassifications. A smaller CCC value makes the SVM more tolerant to misclassifications (including outliers), while a larger CCC focuses on classifying all points correctly, which may lead to overfitting if outliers are present.

4. Kernel Trick:

- Logistic Regression: It operates in the original feature space and does not inherently handle non-linear relationships unless transformed explicitly.
- Support Vector Machines (SVM): SVMs can employ kernel functions to transform the input space, allowing for more flexibility in capturing complex relationships. This capability can also help mitigate the influence of outliers, as SVM can find a more suitable decision boundary in transformed spaces.

Q4. Why don’t we use Mean Squared Error (MSE) as a cost function in logistic regression?

Answer: While mean squared error (MSE) is commonly used in linear regression, it's not the ideal choice for logistic regression. The primary reason for this is the nature of the logistic sigmoid function, which maps values to a range of 0 to 1.

Here's why MSE is not suitable for logistic regression:

- Non-convexity: When used with the logistic sigmoid function, MSE results in a non-convex optimization problem. This means there might be multiple local minima, making it difficult for optimization algorithms to find the global minimum, which corresponds to the optimal model parameters.
- Sensitivity to outliers: MSE is sensitive to outliers, as the squared difference between the predicted and actual values can become very large, disproportionately affecting the overall cost. In logistic regression, outliers can significantly impact the model's performance.

Instead, we typically use cross-entropy loss as the cost function in logistic regression. Cross-entropy is specifically designed for classification problems and addresses the shortcomings of MSE. It is convex, making optimization easier, and is less sensitive to outliers.

In summary, while MSE might seem like a straightforward choice, its limitations in the context of logistic regression make it unsuitable. Cross-entropy is a more appropriate and effective cost function for this task.

Q5. Can you explain the difference between the softmax and sigmoid functions?

Answer: Both softmax and sigmoid functions are commonly used in machine learning, particularly in classification tasks. However, they serve different purposes and have distinct characteristics.

Sigmoid Function:

- Output range: Maps values to a range of 0 to 1.
- Purpose: Often used in binary classification problems, where the output represents the probability of belonging to one of two classes.

- Formula: σ(x) = 1 / (1 + e^(-x))

Softmax Function:

- Output range: Maps a vector of values to a probability distribution, where each element represents the probability of belonging to a corresponding class.
- Purpose: Used in multi-class classification problems, where the output represents the probability distribution over all possible classes.

Formula: softmax(x)_i = e^(x_i) / Σ(e^(x_j))

Key Differences:

- Output: Sigmoid produces a single value, while softmax produces a vector of probabilities.
- Purpose: Sigmoid is suitable for binary classification, while softmax is better suited for multi-class classification.
- Normalization: Softmax ensures that the sum of the output probabilities equals 1, representing a valid probability distribution. Sigmoid does not have this property.

In summary, while both functions are used in classification tasks, the sigmoid function is appropriate for binary classification, and the softmax function is better suited for multi-class classification problems where a probability distribution over all classes is desired.

Q6. Why is the binary cross-entropy loss function convex in logistic regression?

Answer: The binary cross-entropy loss function is convex in logistic regression due to the following properties:

- Convexity of the logistic sigmoid function: The logistic sigmoid function, σ(x) = 1 / (1 + e^(-x)), is itself a convex function. This means that any line segment connecting two points on the curve lies above the curve.
- Composition of convex functions: The binary cross-entropy loss function can be expressed as a composition of convex functions:
    - The negative log function, -log(x), is convex for x > 0.
    - The logistic sigmoid function, σ(x), is convex.

- The composition of two convex functions is also convex. Therefore, the binary cross-entropy loss function, which is a composition of the negative log function and the logistic sigmoid function, is convex.
- Convexity of the sum: The binary cross-entropy loss function is typically calculated as the sum of the cross-entropy losses for individual data points. The sum of convex functions is also convex.

In summary, the convexity of the binary cross-entropy loss function in logistic regression is a result of the convexity of the logistic sigmoid function, the composition of convex functions, and the convexity of the sum. This convexity property is crucial for optimization algorithms to find the global minimum of the loss function, ensuring that the model converges to the optimal solution.

Q7. Can you explain the vectorized implementation of logistic regression?

Answer: Understanding Vectorization

Vectorization is a technique in numerical computing that involves performing operations on entire arrays or matrices at once, rather than element-by-element. This can significantly improve computational efficiency, especially for large datasets.

Logistic Regression Equation

The logistic regression hypothesis function is given by:

$h_\theta(x) = g(\theta^T * x)$

where:

- $g(z) = 1 / (1 + e^{(-z)})$ is the sigmoid function.
- θ is the parameter vector.
- x is the input feature vector.

Vectorizing the Hypothesis Function

- Represent data in matrices:
    - X: A matrix where each row represents a training example, and each column represents a feature.
    - y: A column vector containing the corresponding labels (0 or 1).

    - θ: A column vector containing the parameters.
- Calculate the hypothesis function:
    - h = g(X * θ)
    - This matrix multiplication computes the hypothesis for all training examples in a single operation.
- Vectorizing the Cost Function
    - The cost function for logistic regression is:
    - J(θ) = (1/m) * Σ(y(i) * log(h_θ(x(i))) + (1 - y(i)) * log(1 - h_θ(x(i))))
- Calculate the log terms:
    - log_h = log(h)
    - log_1_minus_h = log(1 - h)
- Calculate the cost:
    - J = (1/m) * sum((y * log_h) + ((1 - y) * log_1_minus_h))
- Vectorizing the Gradient
    - The gradient of the cost function is:

      ∂J/∂θ_j = (1/m) * Σ((h_θ(x(i)) - y(i)) * x_j(i))

1. Calculate the difference:
    1.1. delta = h - y
2. Calculate the gradient:
    2.1. grad = (1/m) * (X^T * delta)

Benefits of Vectorization

- Efficiency: Matrix operations are often optimized in numerical libraries, leading to significant speed improvements.
- Readability: Vectorized code is often more concise and easier to understand.
- Consistency: It ensures consistent calculations across all training examples.

By vectorizing the logistic regression implementation, you can significantly improve its computational efficiency and make it suitable for handling large datasets.

Q8. If you know there are outliers in your data, would you still use logistic regression?

Answer: Yes, logistic regression can still be used with outliers, but it's important to be aware of their potential impact and take appropriate measures.

Here's why:

- Robustness: Logistic regression is relatively robust to outliers compared to some other algorithms. The sigmoid function helps to dampen the influence of extreme values.
- Outlier detection: You can use techniques like standard deviation or interquartile range (IQR) to identify outliers and decide how to handle them.

However, there are some considerations:

- Impact on model performance: Outliers can still affect the model's performance, especially if they are numerous or influential.
- Data cleaning: If outliers are deemed to be errors or anomalies, it might be beneficial to clean the data by removing or correcting them.
- Robust regression techniques: For extreme cases of outliers, consider using robust regression techniques like Huber loss or least absolute deviations (LAD) regression.

In conclusion, while logistic regression can handle outliers to some extent, it's essential to assess their impact on your model and consider appropriate strategies to mitigate their influence. If outliers are particularly problematic, exploring robust regression techniques might be a worthwhile option.

Q9. When would you choose Support Vector Machines (SVM) over Logistic Regression, and vice versa?

Answer: SVM (Support Vector Machine) and Logistic Regression are both powerful classification algorithms, but they have different strengths and weaknesses. The best choice for a particular problem depends on several factors:

SVM:

Advantages:

- Handles high-dimensional data well.
- Effective with small datasets.
- Can handle non-linear decision boundaries.
- Can be used for both classification and regression.

Disadvantages:

- Can be computationally expensive for large datasets.
- Choosing the right kernel can be challenging.
- Less interpretable than logistic regression.

Logistic Regression:

Advantages:

- Produces probabilistic outputs, making it easier to interpret.
- Faster to train and predict than SVM for large datasets.
- Can be easily regularized to prevent overfitting.

Disadvantages:

- Assumes linear separability between classes.
- May not perform well with complex, non-linear decision boundaries.

When to use SVM:

- Small datasets: SVM can perform well with limited data.
- Non-linear decision boundaries: If the data is not linearly separable, SVM can handle complex patterns.
- High-dimensional data: SVM can handle features with many dimensions.
- Need for generalization: SVM is good at generalization, especially with regularization techniques.

When to use Logistic Regression:

- Large datasets: Logistic regression is faster and more efficient for large datasets.
- Linear separability: If the data is linearly separable, logistic regression is a good choice.
- Interpretability: Logistic regression provides probabilistic outputs that are easier to interpret.
- Need for regularization: Logistic regression can be easily regularized to prevent overfitting.

Q10. How would you compare Naive Bayes and Logistic Regression for solving classification problems?

Answer: Both Naive Bayes and Logistic Regression are popular algorithms for classification tasks, but they have different assumptions and strengths. Here's a comparison:

Naive Bayes:

- Assumption of Independence: Naive Bayes assumes that features are conditionally independent given the class label, which simplifies computations. This assumption often holds in practice for text classification (e.g., spam detection).
- Performance with Small Datasets: It performs well with small datasets and can be effective even when the independence assumption is violated to some extent.
- Speed and Efficiency: Naive Bayes is computationally efficient, both in training and prediction, making it suitable for real-time applications.
- Works Well with Categorical Data: It's particularly effective for categorical data, using probability distributions (Gaussian, Multinomial, Bernoulli) depending on the data type.

Logistic Regression:

- No Independence Assumption: Logistic Regression does not assume independence among features, making it more flexible in capturing relationships between them.
- Interpretability: It provides coefficients that can be easily interpreted, offering insights into how each feature affects the prediction.
- Linear Decision Boundary: Logistic Regression is best for linearly separable data. It can struggle with complex decision boundaries unless transformed features are used.
- Requires More Data: Logistic Regression may require larger datasets to achieve stable estimates, especially as the number of features increases.

In summary, I would choose Naive Bayes for simpler, fast classification tasks, especially with categorical data or text. Logistic Regression would be preferred for its interpretability and flexibility in handling continuous data without the independence assumption. The choice depends on the data characteristics and the specific problem context.

Q11. Can logistic regression be used for an imbalanced classification problem?

Answer: Yes, logistic regression can be used for imbalanced classification problems. However, it's important to be aware of the potential challenges and take appropriate measures to address them.

Challenges of Imbalanced Data in Logistic Regression:

- Bias towards the majority class: Logistic regression, by default, tends to be biased towards the majority class in imbalanced datasets. This is because the algorithm aims to minimize the overall error, which can lead to neglecting the minority class.

Strategies to Address Imbalanced Data:

1. Oversampling:
    - Random oversampling: Replicates samples from the minority class to balance the dataset.
    - SMOTE (Synthetic Minority Over-sampling Technique): Generates new synthetic samples for the minority class based on existing ones.
2. Undersampling:
    - Randomly removes samples from the majority class to balance the dataset.
    - Tomek links: Removes pairs of samples from different classes that are very close to each other.
3. Class weighting:
    - Assigns higher weights to the minority class during training to give it more importance.
4. Ensemble methods:
    - Combine multiple models, such as random forests or gradient boosting, to improve performance on imbalanced datasets.
5. Cost-sensitive learning:
    - Adjust the cost function to penalize misclassification of the minority class more heavily.

Choosing the Right Strategy:

The best strategy depends on the specific characteristics of your dataset and the desired trade-offs between accuracy and class balance. Experimentation with different techniques is often necessary to find the most effective approach.

In conclusion, while logistic regression can be used for imbalanced classification problems, it's essential to be aware of the potential biases and take appropriate measures to address them. By employing techniques like oversampling, undersampling, class weighting, or ensemble methods, you can improve the performance of logistic regression on imbalanced datasets.

Bias Variance

Q1. How can you identify a High Bias Model? How can you fix it?

Answer : Bias and variance contribute to a model's predictive power and can be balanced through various methods.

1. High training error
2. Validation error or test error is the same as training error

To fix a High Bias model, you can:

1. Add more input features
2. Add more complexity by introducing polynomial features
3. Decrease the regularization term

Q2. How do bias and variance contribute to the overall error in a predictive model?

Answer : Bias and variance contribute to a model's predictive power and can be balanced through various methods.

Architectural Impacts: Bias & Variance

- Bias: Represents the model's inability to capture complex relationships in the data, leading to underfitting.
- Variance: Reflects the model's sensitivity to small fluctuations or noise in the training data, often causing overfitting.

The Bias-Variance Tradeoff

The bias-variance decomposition framework aids in understanding prediction errors and managing model complexity.
The expected error of a learning model can be represented as the sum of three distinct components:

Expected Error

$$E(y - f(x))^2 = Var(f(x) + Bias^2(f(x)) + Var(\epsilon)$$

Where:

- y is the true output.
- f(x) denotes the model's prediction for input x.
- ϵ represents the error term, assumed to be independent of x.
  The three components contributing to error are:
  i. Noise variance: The irreducible error present in all models.
  ii. Bias^2: The degree to which the model doesn't capture true relationships in the data.
  iii. Variance: The extent to which the model's predictions vary across different training datasets.

Code

```
import numpy as np

# True output
y_true = np.array([1, 2, 3, 4, 5])

# Mean of the true output
y_mean = np.mean(y_true)

# Predictions from a model
y_pred = np.array([1, 3, 3, 5, 5])

# Calculate total variance
total_variance = np.var(y_true)

# Calculate variance in predictions
pred_variance = np.var(y_pred)

# Calculate bias squared
bias_squared = np.mean((y_pred - y_mean) ** 2)

# Calculate noise variance
noise_variance = total_variance - pred_variance - bias_squared
```

```
# Output the variances and squared bias along with noise
print("Variance contribution from the predictions: ", pred_variance)
print("Squared bias contribution from the predictions: ",
bias_squared)
print("Noise variance contribution: ", noise_variance)
```

Q3. Can Machine models overcome underfitting on biased data and overfitting on data with variance? Does this guarantee correct results?

Answer : Yes, they can. Underfitting can be overcome by utilizing ML models with a greater emphasis on the features – increasing the number of features or placing greater weight on the features at play (using higher degree polynomials, for example.)
As for overfitting, the reverse can be done to eradicate it.
This does guarantee plausible results in real life since they still may be based on data that has not been collected with the proper technique.

Q4. Show Bias Variance Trade-Off in Linear Regression

Answer :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Set up data
np.random.seed(0)
X = np.linspace(0, 10, 100)
y = 2*X + np.random.normal(0, 1, 100)

# Instantiate models of varying complexity
models = {
    'Underfit': LinearRegression(),  # Normal Linear Regression
    'Optimal': LinearRegression(fit_intercept=False), # If there is no bias
    'Overfit': LinearRegression(copy_X=True) # Bias
}

# Train models and calculate error metrics
train_sizes, train_scores, test_scores = learning_curve(models[-1],
X.reshape(-1, 1), y, cv=5)
```

```
train_errors, test_errors = [], []
for key, model in models.items():
    model.fit(X.reshape(-1, 1), y)
    train_pred, test_pred = model.predict(X.reshape(-1, 1)),
model.predict(X.reshape(-1, 1))
    train_errors.append(mean_squared_error(y, train_pred))
    test_errors.append(mean_squared_error(y, test_pred))

# Visualize the data
fig, ax = plt.subplots(1, 1, figsize=(5,3))
ax.plot(train_sizes, train_scores, 'o-', color='r', label='Training Set')
ax.plot(train_sizes, test_scores, 'o-', color='g', label='Testing Set')
ax.set_xlabel('Model Complexity')
ax.set_ylabel('Performance')
ax.set_title('Learning Curve')
ax.legend()
plt.show()

# Print error metrics
print("Training Errors:\n", train_errors, '\n')
print("Testing Errors:\n", test_errors)
```

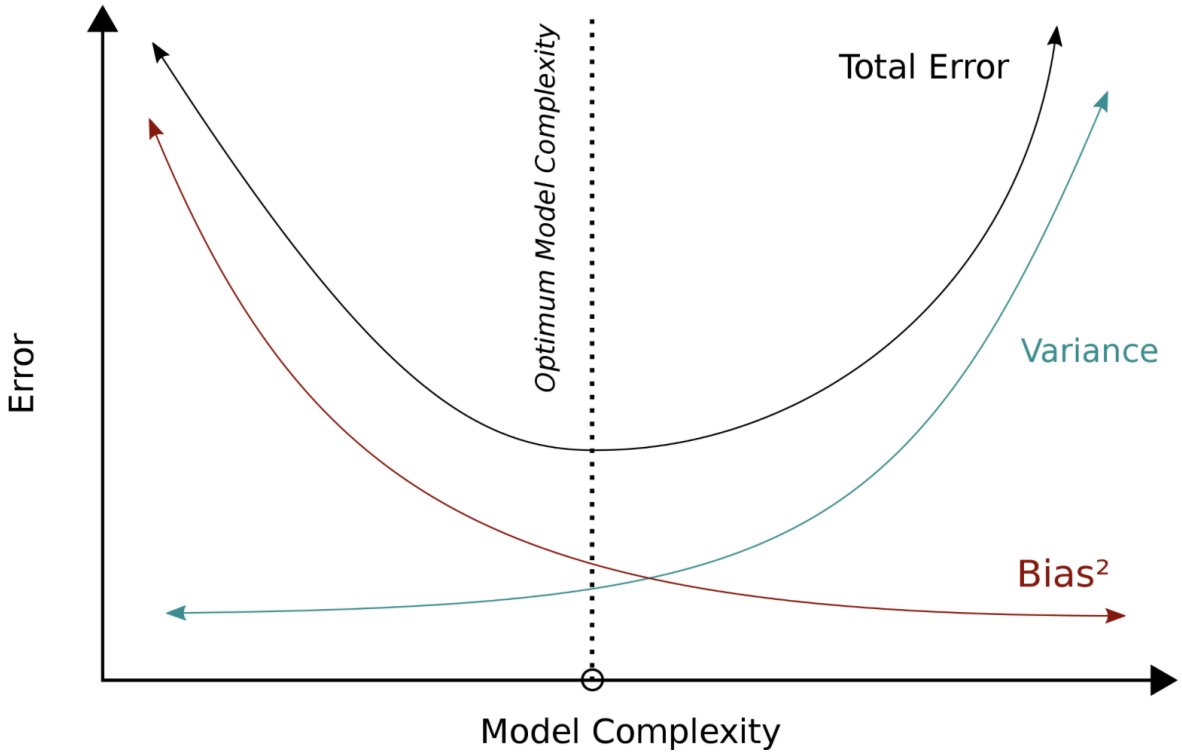
Q5. Why is it impossible to simultaneously minimize both bias and variance?

Answer : Attempting to minimize both bias and variance is an example of the Bias-Variance Dilemma, which stems from the inherent trade-off between these two sources of error.

Bias-Variance Dilemma

The Bias-Variance Dilemma asserts that improving a model's fit to the training data often compromises its generalization to unseen data, because reducing one type of error (E.g., bias) typically leads to an increase in the other (E.g., variance).

Visual Representation



Mathematical Representation
The mean squared error (MSE) is the sum of bias, variance, and irreducible error:
MSE=E[(θ' − θ)^2]=bias2+variance+irreducible error
Where θ' is the estimated parameter, θ is the true parameter, and E denotes the expected value.

Mathematical Detail
Bias: Represents the errors introduced by approximating a real-life problem, such as oversimplified assumptions. It quantifies the difference between the model's expected prediction and the true value. Minimizing bias involves creating a more complex model or using more relevant features, which could lead to overfitting.
Bias=E[θ']−θ
Variance: Captures the model's sensitivity to small fluctuations in the training data. A high variance model is highly sensitive, leading to overfitting. Reducing variance usually involves simplifying the model, which can lead to higher bias.
Variance=E[(θ'−E[θ'])^2]

Irreducible Error: This error term arises from noise in the data that is beyond the control of the model. It represents a lower limit on the obtainable error rate and cannot be reduced.
Irreducible Error=σ^2
Unified Approach
In statistical learning and state-of-the-art Machine Learning, models aim to strike a balance between bias and variance by overall error minimization. Techniques like cross-validation, regularization, and ensemble methods help manage this bias-variance trade-off, yielding models that can generalize to new data effectively.

Q6. How would you diagnose bias and variance issues using learning curves?

Answer : One of the most effective ways to diagnose both bias and variance issues in a machine learning model is through the use of Learning Curves.
Learning Curves are graphs that show how a model's performance on both the training data and the testing data changes as the size of the training set increases.
Key Indicators from Learning Curves

- Training Set Error: The performance of the model on the training set.
- Validation Set (or Test Set) Error: The performance on a separate dataset, usually not seen by the model during training.
- Gap between Training and Validation Errors: This gap is a key indicator of variance.
- Overall Level of Error: The absolute error on both the training and validation sets indicates the bias.

Visual Cues for Bias and Variance
High Variance
Visual Clues: Large gap between training and validation error; both errors remain high.
Cause: The model is overly complex and tries to fit the noise in the training data, leading to poor generalization.
High Bias
Visual Clues: Small gap between training and validation errors, but they are both high.
Cause: The model is too simple and is unable to capture the underlying patterns in the data.
Balancing Bias and Variance

- Visual Clues: Errors converge to a low value, and there's a small, consistent gap between the two curves.
- Desirable Scenario: The model manages to capture the main patterns in the data without overfitting to noise.
- Cross-Verification
  It's crucial to validate your conclusions about bias and variance stemming from learning curves using other metrics, such as area under the receiver operating

characteristic curve (AUC-ROC), precision-recall curves, or by employing k-fold cross-validation.

Clustering

Q1. Can you explain the difference between supervised and unsupervised learning with examples of where K-Means Clustering fits in?

Answer : K-Means Clustering falls under unsupervised learning, in contrast to supervised learning methods like Decision Trees and Random Forest.

Unsupervised Learning: Discovering Patterns

In unsupervised learning, the algorithm doesn't rely on labeled data and operates by identifying commonalities or patterns in the dataset. This method is more about exploration and understanding.

K-Means Clustering: Grouping Data

K-Means is a partition-based method, clustering data by minimizing the Euclidean distance between each point and the centroid of its associated cluster.

Example: A real-world application is in customer segmentation, where similar customers are grouped together based on their shopping behavior.

Supervised Learning: Assigning Labels

Supervised learning is all about teaching a model how to map input to output based on example data. The model then uses that knowledge to predict the correct output when presented with new input.

Decision Trees: Classification and Regression

A Decision Tree sequentially segments the data in a tree-like structure. At each node, the tree makes a decision that best separates the data based on certain features.

Example: Decision Trees can be used in medicine to diagnose patients based on symptoms.

Random Forest: Ensemble Learning

Random Forest is an ensemble of Decision Trees. It constructs several trees through bootstrapping, each considering a subset of features, and combines their predictions through voting (for classification) or averaging (for regression).

Example: A practical application is in predicting customer churn for businesses.

Q2. What are *centroids* in the context of *K-Means*?

Answer :In k-Means clustering, centroids represent data points that act as the center of clusters. Each cluster is defined by its corresponding centroid, and the goal of the algorithm is to minimize intra-cluster distances by optimizing these centroids.

Role of Centroids

- Cluster Assignment: Each data point is associated with the closest centroid, effectively linking it to a specific cluster.
- Initial Centroid Selection: Starting with an initial guess, the algorithm iteratively updates these points. Convergence occurs when the centroids no longer shift substantially.
- Model Representation: The optimized centroids, alongside the assigned data points, define the k-Means model. This model can make inferences about future, unseen data by assigning them to the nearest centroids.

Centroid Calculation

Mathematically, the centroid of a cluster Ck having nk data points in a d-dimensional space is given by the mean of the data points in that cluster:

$$\mathbf{c}_k = \frac{1}{n_k} \sum_{\mathbf{x} \in C_k} \mathbf{x}$$

The algorithm aims to find centroids that minimize the within-cluster sum of squares (WCSS), which is also known as inertia. This is measured by:

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \mathbf{c}_k\|^2$$

The smaller this value, the better the clustering.

Q3. What are some methods for *initializing* the *centroids* in *K-Means Clustering*?

Answer : K-Means Clustering efficiency rests on strong initial centroid selection. Incorrect initial seeding might lead to suboptimal cluster separation or slow convergence. Here are the common centroid initialization methods.

K-Means++

K-Means++ enhances the random approach by probabilistically selecting the initial centroids. This initiative lessens the likelihood of starting with close or outlier centroids.

1. Algorithm:
    - Choose the first centroid randomly from the dataset.
    - For every next centroid, pick a sample with a likelihood of being selected proportional to its squared distance from the closest centroid.
    - Keep repeating this procedure until all centroids are chosen.
2. Advantages:
    - Suited for large datasets.
    - Still relatively efficient even when k (the number of clusters) is not small.
3. Code Example: Here is the Python code:

```
from sklearn.cluster import KMeans
```

4. kmeans = KMeans(n_clusters=4, init='k-means++')

K-Means

K-Means is the classic approach where the initial centroids are randomly picked from the dataset.

1. Random Sampling:
    - Select k observations randomly from the dataset as initial centroids.
2. Advantages:
    - Quick and easy to implement.
    - Suitable for small datasets.
3. Code Example: Here is the Python code:

```
from sklearn.cluster import KMeans
```

4. kmeans = KMeans(n_clusters=4, init='random')

Q4. How does the choice of ‘k’ impact the K-means algorithm?

Answer : The choice of ‘k’ influences the number of clusters identified by the algorithm. Selecting an optimal ‘k’ is crucial; too few or too many clusters can result in misleading or overly detailed groupings.

Q5. What is the silhouette score, and how is it used in evaluating clustering results?

Answer: The silhouette score measures how well-defined the clusters are in a clustering result. It ranges from -1 to 1, where a higher value indicates better-defined clusters.

Q6. What is the difference between K-means and hierarchical clustering?

Answer: K-means is a partitioning algorithm that assigns each data point to a single cluster, while hierarchical clustering creates a tree-like structure where data points can belong to multiple clusters at different levels.

Q7. What is the elbow method, and how is it used in determining the optimal number of clusters?

Answer: The elbow method involves plotting the cost (or inertia) of K-means clustering for different values of 'k' and identifying the point where the rate of decrease sharply changes, resembling an elbow. This point is considered the optimal number of clusters.

Q8. How does the Affinity Propagation algorithm work?

Answer: Affinity Propagation uses a message-passing approach to let data points vote on the most suitable exemplar, which represents the cluster. It iteratively refines these votes to converge on the final clusters.

Q9. Explain the concept of silhouette width and how it is calculated.

Answer: Silhouette width measures how similar an object is to its own cluster compared to other clusters.
It ranges from -1 to 1, with higher values indicating better-defined clusters.
It is calculated as (b — a) / max(a, b), where 'a' is the average distance within the cluster, and 'b' is the average distance to the nearest cluster.

Q10. How does the Gaussian Mixture Model (GMM) differ from K-means?

Answer: While K-means assumes that clusters are spherical and assigns data points to the nearest cluster, GMM models clusters as ellipses and assigns probabilities to data points belonging to different clusters.

Q11. How does the DBSCAN algorithm handle clusters of different shapes and sizes?

Answer: DBSCAN is capable of identifying clusters of different shapes and sizes since it defines

clusters based on local density. It can handle irregularly shaped clusters and is less sensitive to outliers.

Q12. Explain the concept of medoid in clustering.

Answer: A medoid is a representative point within a cluster, minimizing the average dissimilarity to all other points in the cluster. Unlike a centroid, a medoid is an actual data point.

Q13. How do you handle categorical variables in clustering algorithms?

Answer: Handling categorical variables involves converting them into a numerical format, often through techniques like one-hot encoding. Some clustering algorithms, like K-prototypes, are designed to handle mixed numerical and categorical data.

Q14. What is the difference between k-means and hierarchical clustering?

Answer: K-means clustering partitions the dataset into a predefined number of clusters (k) by minimizing the within-cluster variance, while hierarchical clustering builds a hierarchy of clusters by recursively merging or splitting clusters based on similarity or dissimilarity measures.

Q15. How do you determine the optimal number of clusters in a clustering algorithm?

Answer: The optimal number of clusters can be determined using techniques like the elbow method, silhouette analysis, or the gap statistic. These methods aim to find the point where adding more clusters does not significantly improve the clustering quality or where the silhouette score is maximized.

Q16. What are some applications of unsupervised learning in real-world scenarios?

Answer: Some applications of unsupervised learning include customer segmentation for targeted marketing, anomaly detection in cybersecurity, topic modeling for text analysis, image clustering for visual content organization, and recommendation systems for personalized content delivery.

Q17. What is the purpose of feature scaling in machine learning?

Answer: Feature scaling ensures that all features contribute equally to the model training process by scaling them to a similar range. Common scaling techniques include min-max scaling and standardization (Z-score normalization).

Q18. How do you handle skewed distributions in features?

Answer: Skewed distributions can be transformed using techniques like logarithmic transformation, square root transformation, or Box-Cox transformation to make the distribution more symmetrical and improve model performance, especially for algorithms that assume normality.

Q19. What is the difference between bagging and boosting?

Answer: Bagging (Bootstrap Aggregating) and boosting are ensemble learning techniques used to improve model performance by combining multiple base learners. Bagging trains each base learner independently on different subsets of the training data, while boosting focuses on training base learners sequentially, giving more weight to misclassified instances.

Q20. Explain the working principle of support vector machines (SVM).

Answer: Support Vector Machines (SVM) is a supervised learning algorithm used for classification and regression tasks. It works by finding the hyperplane that best separates the data points into different classes while maximizing the margin, which is the distance between the hyperplane and the nearest data points from each class.

Q21. Explain Kernel SVM.

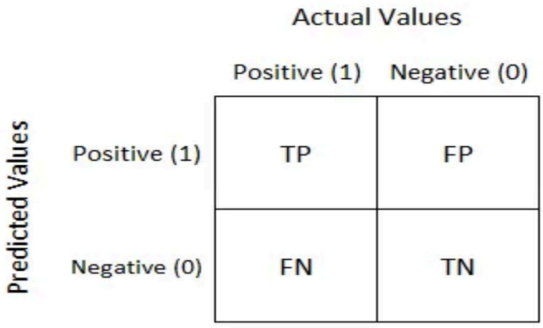
Answer: Kernel SVM stands for Kernel Support Vector Machine. In SVM, a kernel is a function that aids in problem-solving. They provide shortcuts to help you avoid doing complicated math. The amazing thing about kernel is that it allows us to go to higher dimensions and execute smooth computations. Kernel SVMs can work with a variety of kernel functions, including linear, polynomial, and radial basis function (RBF) kernels, among others.

AUC - ROC

Q1. What is a confusion matrix?

Answer: A confusion matrix is another model evaluation technique which we use for classification problems. In this technique we make a NxN matrix, where N is the number of distinct classes to be predicted, so for a binary classification problem N=2. We have actual values on the X-axis and Predicted values on the Y-axis. Model evaluation is done based on

some metrics that can be generated from this, like specificity, sensitivity, precision and recall.



Q2. How do we define sensitivity, specificity and precision?

Answer: Sensitivity, recall, power or true positive rate is the ratio of True positives to total number of positives. For a given classification problem, it tells us the proportion of positives that are correctly classified from all the positives.

$$Sensitivity = \frac{TP}{TP + FN}$$

Specificity or true negative rate is the ratio of true negatives to total negatives. It tells us the proportion of correctly classified negatives from all negatives.

$$Specificity = \frac{TN}{TN + FP}$$

Precision is the proportion of True Positives to all cases Predicted as Positives

$$Precision = \frac{TP}{TP + FP}$$

Q3. Explain Naive Bayes' algorithm

Answer: Naive Bayes' is a classification algorithm that uses Bayes' theorem to categorize instance to a specific class. It assumes independence between the predictors which is a major assumption given that it's almost impossible that a set of predictors are completely independent. It works well when the predictors are categorical.

$$P(Class|Predictor) = \frac{P(Predictor|Class) * P(Class)}{P(Predictor)}$$

$$PosteriorProbability = \frac{Likelihood * ClassPriorProbability}{PredictorPriorProbability}$$

This should be a good place to stop. In this part II we will be covering some of the most commonly asked interview question on different ML algorithms.

Q4. What are the two components of Bayesian logic?

Answer: Two components of Bayesian logic are as follows:

Prior distribution: This is the information or beliefs that you have about the world before observing any new data. Prior knowledge is typically represented as a probability distribution over possible states of the world.
Likelihood principle: This is a function that describes the probability of observing some data given a particular state of the world. The likelihood function is typically derived from scientific theories, empirical observations, or expert opinions.

Q5. What is an F-score?

Answer: F score is a measure to test accuracy of a binary classification. It is the harmonic mean of precision and recall. The F-score takes on a value between 0 and 1 and higher the F-score, better the power of the classification model.

$$F - score = 2 * (\frac{Precision * Recall}{Precision + Recall})$$

Q6. When do we use ROC and AUC? What do they stand for? Why would we use an ROC curve?

Answer: The ROC curve plots the True Positive Rate (Sensitivity) vs. False Positive Rate (1-Specificity). ROC curves are used for parameter tuning. Let's say we have a logistic regression model and we don't know what threshold to use for classifying True Positives and True Negatives, then in this case generating a confusion matrix for each threshold can be a cumbersome process to decide which threshold to use. So we make the ROC curve to decide which threshold to use based on the true positive and false positive rate that works best for us. We use AUC to decide which model is better. The model that has the maximum Area Under Curve is the best model.

Q7. What is the difference between odds and probability? How are each defined? How are log(odds) related to logistic regression?

Answer: Odds are the ratio of something happening to that something not happening, whereas probability is the ratio of something happening to the total number of possibilities.

The probability in logistic regression is modeled as $P(X) = \frac{e^{\beta 0 + \beta 1 * x}}{1 + e^{\beta 0 + \beta 1 * x}}$

Now if we transform this equation to make it linear in X, we get something like $log(\frac{P(X)}{1 - P(X)}) = \beta 0 + \beta 1 * x$ where the left side represents the log(odds).

PCA

Q1. Is PCA used in feature selection?

Answer:
No, PCA is not typically used for feature selection. Instead, it is used in feature engineering to create new features. PCA transforms the original features into a new set of uncorrelated components, capturing the most variance in the data. These components are linear combinations of the original features, so it doesn't directly "select" existing features but rather forms new ones based on the data's variance.

Q2. Given the following small dataset:

X1	X2
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0

How would you apply PCA to this dataset? Explain the steps and provide the code.

Answer:
Steps to Apply PCA:
Standardize the Data: PCA requires the data to be standardized, so the mean of each feature is 0 and the variance is 1.
Fit PCA: Compute the covariance matrix and extract eigenvalues and eigenvectors to identify the principal components.
Transform the Data: Project the original data onto the new principal components.
Transformed Data (Principal Components):
[-0.48592434,2.25837321,-0.25339487,-0.83020891,-0.6888449]

CODE:

```
data = np.array([[2.5, 2.4], [0.5, 0.7], [2.2, 2.9], [1.9, 2.2], [3.1,
3.0]])
# Standardize the data
stand_data = (data - np.mean(data,axis=0))/np.std(data,axis=0)
# Covariance Matrix
n = stand_data.shape[0]
m = stand_data.shape[1]
cov_matrix = np.zeros((m,m))

for i in range(m):
    for j in range(m):
        cov_matrix[i,j] =
(1/(n-1))*(np.sum(stand_data[:,i]*stand_data[:,j]))

# Eigenvalues and Eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Sort the Eigenvalues and Eigenvectors
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvectors_sorted = eigenvectors[:, sorted_indices]

# User input for number of principal components selection
k=1

# Select top k eigenvectors i.e., Principal Components
principal_components = eigenvectors_sorted[:,:k]

# Projecting the data onto selected principal components
transformed_data = stand_data.dot(principal_components)
```

Errors and Cross-Validation

Q1. Why in classification tasks squared error is not a suitable loss function?

Answer:
1. Non-probabilistic outputs: In classification, the goal is to predict discrete class labels, often based on the probability of each class. Squared error assumes continuous outputs, which makes it more suited for regression tasks where predictions are numerical. It fails to capture the probabilistic nature of classification.
2. Insensitive to misclassification: Squared error treats all differences between predicted and actual values equally. In classification, the concern is whether the predicted class is correct or not, rather than the magnitude of the error.

Q2. Why does the loss function (example: - sum of squared errors, mean squared error) in linear regression models typically square the error of the outputs?

Answer:
1. To make the error positive so that the negative error doesn't cancel out the positive error in the expression.
2. Squaring the errors also makes sure that large errors are more penalized than the smaller errors which helps the model focus on minimizing large errors in its predictions.

Q3. Does increasing the model complexity always result in a reduction in error for the training set?

Answer: Yes, increasing the model complexity always results in a reduction in error for the training set but the model that has high complexity may not be ideal for the task as even though they have low bias, they have high variance if the model is not trained on a large set so it may not perform better on the validation set.

Q4. Suppose your dataset has significant class imbalance (for classification problems), what are the challenges that can arise during cross-validation, and how would you address them without modifying the dataset?

Answer: Problems arising due to an imbalanced dataset:

1. Algorithms may get biased towards the majority class and thus tend to predict output as the majority class.
2. Minority class observations may look like noise to the model and be ignored by it.
3. An imbalanced dataset gives misleading accuracy scores.

Ways to address them:

1. Using different evaluation metrics as classifier accuracy is suitable for balanced classes but less effective for imbalanced classes. F1 scores is the preferred metric for the imbalanced datasets as it decreases when the classifier predicts the minority class incorrectly.
2. Resampling:
   Randomly removing rows from the majority class to align with the minority class.

Q5. How do you combine cross-validation with hyperparameter tuning and what risks will you face while performing this process?

Answer: Decision trees are semi-parametric models that bridge the gap between them.
They exhibit a parametric nature as once they are constructed; they have a finite and fixed no of parameters, but the form of the decision tree is heavily influenced by the training data as the structure—the depth of the tree, the number of splits, and the variables chosen at each split—depends entirely on the input data that is it is data-driven.

Q6. Are decision trees parametric or non-parametric models?

Answer: Cross-validation can be used for both hyperparameter tuning and estimating the generalization performance of the model.
This method involves two levels of cross-validation:
An inner CV for parameter search and an outer CV for the best model selection.
The outer CV loop defines the dataset splits that the inner CV loop uses to find the best set of hyperparameters by performing.

Risks Faced: -

Using the same cross-validation for both purposes simultaneously can lead to increased bias especially when the dataset is small.

Decision Trees, KNN, SVM, Random Forest

Q1. Are decision trees parametric or non-parametric models?

Answer: Decision trees are semi-parametric models that bridge the gap between them. They exhibit a parametric nature as once they are constructed; they have a finite and fixed no of parameters, but the form of the decision tree is heavily influenced by the training data as the structure—the depth of the tree, the number of splits, and the variables chosen at each split—depends entirely on the input data that is it is data-driven.

Q2. Why do we even apply methods of decision trees, linear regression, etc. when we can just apply neural network solutions to all the problems?

Answer: The main reasons we apply these methods are: -
1. They are fast to implement and have few or no hyper-parameters to tune.
2. They often work as well or better than more complicated methods.
3. Both can be easier to explain to a human user: decision trees are directly human-interpretable and nearest neighbor methods can justify their decision to some extent by showing a few training examples that the prediction was based on.

Q3. Why can't gradient descent be applied to minimize the error function in regression trees?

Answer: Gradient descent cannot be applied directly to minimize the error function of regression trees because decision trees are built using discrete, non-differentiable operations (such as selecting features and splitting thresholds).

Q4. Suppose we train an SVM multiple times, with different random initializations at the start, still, why do we get the same weights once the training is complete every time?

Answer: SVMs optimize a convex objective function, specifically maximizing the margin between classes while minimizing classification errors. The convex nature of this function ensures that it has one unique solution.

Q5. What is the main advantage of using a decision tree in comparison to more powerful alternatives like neural networks?

Answer: The main advantage of decision trees over neural networks is its simplicity and interpretability as each decision node corresponds to a feature and a decision threshold, making it straightforward to visualize and explain to non-experts. Neural networks, in contrast, function as complex "black boxes" with numerous layers and parameters, making them difficult to interpret.

Q6. In KNN why don’t we take k value as an even number?

Answer: In KNN, the class label of a new data point is determined by a majority vote of its k nearest neighbours. If k is an even number and there are two or more classes, the algorithm may end up with an equal number of neighbours from two different classes, resulting in a tie which further complicates the process, so it is highly discouraged to use k value as an even number.

Q7. Why is the role of distance metric important in K nearest neighbor?

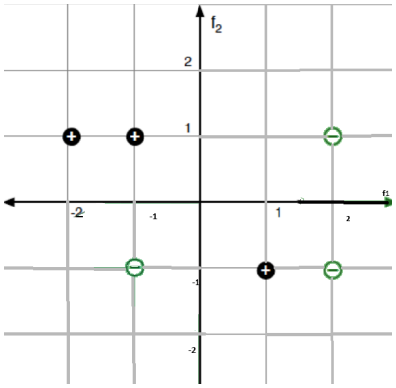
Answer: The role of the distance metric is crucial in K-Nearest Neighbors (KNN) because it determines how the algorithm calculates the "closeness" of data points and, ultimately, which neighbours influence the classification or prediction.
Choosing the appropriate metric that fits the structure of your data can significantly impact the accuracy of the model.

Q8. Why is the role of distance metric important in K nearest neighbor?

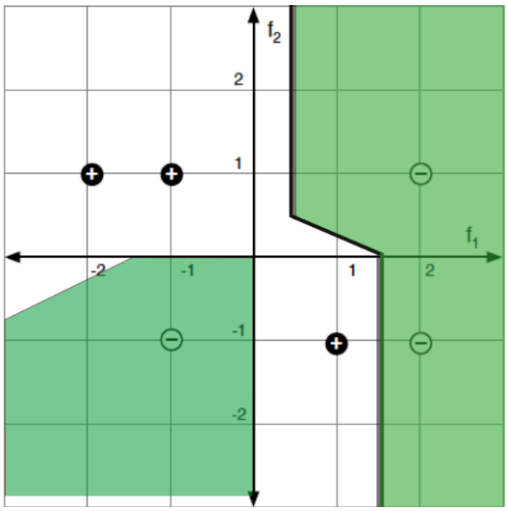
Answer: The role of the distance metric is crucial in K-Nearest Neighbors (KNN) because it determines how the algorithm calculates the "closeness" of data points and, ultimately, which neighbours influence the classification or prediction.
Choosing the appropriate metric that fits the structure of your data can significantly impact the accuracy of the model.

Q9. The nearest neighbour prediction function can be described by dividing the space up into regions whose closest point is each training point.

Describe the following diagram by dividing it into regions:



Answer:



Q9. What's the difference between Random Forest and simply bagging a decision tree?

Answer: Bagging trains multiple decision trees using different bootstrap samples, but each tree uses all features for splits, while Random Forest also trains multiple trees on bootstrap samples, but it randomly selects a subset of features at each split, increasing tree diversity and reducing overfitting. This additional randomness in feature selection makes random forests generally more powerful and robust than the simple bagging of decision trees.

Q10. A distance metric for a k nearest Neighbour is defined as the following:

i) $d(A, B) = |A - B|$ if $A > B$

ii) $d(A, B) = 2 \times |A - B|$ otherwise

Is this a valid metric?

Answer: No, it is not a valid metric as it violates as it is not symmetric that is the distance metric gives different values depending on the order of the points, violating the symmetry property.

Q11. What is the meaning of support vectors in SVM?

Answer: Support vectors are the data points that lie closest to the decision boundary (or hyperplane) and play a crucial role in defining that boundary. These points are critical because they determine the position and orientation of the decision hyperplane that separates the different classes.

Q12. Suppose we add new data entries to a dataset away from the hyperplane. Will there be a change in the position of the hyperplane due to the new entries?

Answer: Gradient descent cannot be applied directly to minimize the error function of regression trees because decision trees are built using discrete, non-differentiable operations (such as selecting features and splitting thresholds).

Q13. Suppose we add new data entries to a dataset away from the hyperplane. Will there be a change in the position of the hyperplane due to the new entries?

Answer: No, if the new data entries are far away from the hyperplane, they will not affect the position of the hyperplane in Support Vector Machines (SVM) as the hyperplane position is entirely dependent on support vectors which are data points that lie closest to the hyperplane.

Q14. How can I adjust a model with high bias and low variance to achieve low bias and high variance?

Answer:
1. Use a more complex model:
Switch to a more expressive model: If you're using a simple model like linear regression (which has high bias), switching to a more complex model like decision trees, random forests, or deep learning can reduce bias but increase variance.
2. Reduce regularization:
Decrease regularization strength: If your model is highly regularized (like Ridge or Lasso regression), reduce the regularization parameter (λ). Regularization is typically used to control variance but decreasing it can reduce bias while increasing variance.
3. Increase the number of features:
Include more features or interaction terms: By adding more features or interaction terms, you allow the model to learn more complex relationships, which can reduce bias but may lead to overfitting (higher variance).
4. Decrease model constraints:
If you are using models with specific constraints (e.g., restricting the maximum depth of a decision tree), relaxing these constraints will reduce bias and increase variance.
5. Increase model capacity:
Add more layers/nodes to neural networks: If using a neural network, increasing the number of hidden layers or neurons will increase the model's capacity to learn complex patterns, reducing bias at the cost of higher variance.

Q15. What are the different types of gradient descent, how do they differ, and when should each be applied?

Answer:
1. Batch Gradient Descent (BGD):
In batch gradient descent, the gradient is computed over the entire training dataset for every update of the model's parameters. This provides a stable and accurate estimate of the gradient.
When to apply: Use batch gradient descent when the dataset is small or when computational resources allow for processing large datasets in a reasonable time frame.

2. Stochastic Gradient Descent (SGD):
In stochastic gradient descent, the gradient is computed, and the model is updated after evaluating each individual training example. This introduces noise to the gradient estimation but allows much faster updates.
When to apply: Use SGD when you have a large dataset and need fast, frequent updates. It's also useful when memory is limited, as it processes one example at a time.

Q16. Why is bootstrapping used in Random Forest, and what would happen if it's not used?

Answer:
1. Why Bootstrapping is Used:
i. Increases Diversity Among Trees: By training each tree on a slightly different subset of the data, Random Forest introduces variability in the decision trees. This helps avoid overfitting and increases the overall generalization ability of the ensemble.

ii. Reduces Variance: A single decision tree is prone to high variance and may overfit the data. By averaging the predictions of multiple trees trained on different datasets, bootstrapping helps reduce the variance of the final model.
iii. Improves Robustness: Since each tree sees a different portion of the data, the final model is more robust to anomalies or outliers, as individual trees might handle them differently.

2. What Happens if Bootstrapping is Not Used:
I. Increased Correlation Between Trees: Without bootstrapping, all trees in the forest would be trained on the same dataset. This makes the trees more similar to each other, leading to highly correlated predictions. This reduces the ensemble's ability to lower variance and make accurate predictions.
ii. Reduced Generalization: The lack of diverse training data for each tree would make the Random Forest act more like a traditional decision tree ensemble, increasing the likelihood of overfitting and reducing the generalization performance.
iii. Lower Model Performance: The strength of Random Forest lies in the fact that it combines many "weak" learners into a strong one. Without bootstrapping, the trees would be less diverse, and the final model would lose the ensemble benefit, likely leading to lower accuracy on unseen data.

Q17. What are the differences between bagging, boosting, and bootstrapping

Answer:
1. Bootstrapping:
Bootstrapping is a sampling technique where multiple subsets of data are created by sampling with replacement from the original dataset. Each sample has the same size as the original dataset, but because of replacement, some data points may appear multiple times, while others may not appear at all.
2. Bagging:
Bagging is an ensemble technique that builds multiple models (typically decision trees) by training them on different bootstrapped datasets (subsets of the data created through bootstrapping). The final prediction is made by averaging the outputs of all models (for regression) or taking a majority vote (for classification).
3. Boosting:
Boosting is another ensemble technique where models are built sequentially. Each new model is trained to correct the mistakes made by the previous ones. Unlike bagging, boosting does not use bootstrapped datasets. Instead, it adjusts the weights of training examples, focusing on those that were misclassified or poorly predicted by previous models.

Q18. What happens to a Random Forest when we increase the depth of the trees?

Answer:
1. Higher Complexity and Overfitting:
Deeper trees allow the model to learn more complex patterns in the training data. This can lead to overfitting, where the trees capture noise and specific details that do not generalize well to new, unseen data.
2. Lower Bias:
Deeper trees are less restricted and can better fit the training data. This reduces bias, allowing the trees to make more accurate predictions on the training set.
3. Increased Training Time:
Deeper trees require more computations since they need to evaluate more splits and branches. This increases the overall training time, especially if there are many features.

Q19. Implement from scratch,

1. Random Forest
2. KNN

Answer:

1.

```
import random
from collections import Counter
import numpy as np

class DecisionTree:
   def __init__(self, max_depth=2):
      self.max_depth = max_depth
      self.tree = None

      def fit(self, X, y):
      self.tree = self._grow_tree(X, y, depth=0)

      def _grow_tree(self, X, y, depth):
      num_samples, num_features = X.shape
      if depth >= self.max_depth or len(set(y)) == 1:
            return Counter(y).most_common(1)[0][0]

      feat_idx = random.randint(0, num_features - 1)
      threshold = np.mean(X[:, feat_idx])
      left_idxs = X[:, feat_idx] < threshold
      right_idxs = X[:, feat_idx] >= threshold

      left = self._grow_tree(X[left_idxs], y[left_idxs], depth + 1)
      right = self._grow_tree(X[right_idxs], y[right_idxs], depth +
1)
      return (feat_idx, threshold, left, right)

      def predict(self, X):
      return np.array([self._traverse_tree(x, self.tree) for x in X])

      def _traverse_tree(self, x, node):
      if not isinstance(node, tuple):
```

```
            return node
        feat_idx, threshold, left, right = node
        if x[feat_idx] < threshold:
            return self._traverse_tree(x, left)
        else:
            return self._traverse_tree(x, right)

class RandomForest:
    def __init__(self, n_trees=10, max_depth=2):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.trees = []

    def bootstrap_sample(self, X, y):
        n_samples = X.shape[0]
        indices = np.random.choice(n_samples, size=n_samples,
replace=True)
        return X[indices], y[indices]

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTree(max_depth=self.max_depth)
            X_samp, y_samp = self.bootstrap_sample(X, y)
            tree.fit(X_samp, y_samp)
            self.trees.append(tree)

    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])

    return np.array([Counter(tree_preds[:, i]).most_common(1)[0][0] for i
    in range(X.shape[0])])
```

2.

```
import numpy as np
from collections import Counter
 def euclidean_distance(x1, x2):
```

```
    return np.sqrt(np.sum((x1 - x2) ** 2))

class KNN:
    def __init__(self, k=3):
    self.k = k

    def bootstrap_sample(self, X, y):
    n_samples = X.shape[0]
    indices = np.random.choice(n_samples, size=n_samples, replace=True)
    return X[indices], y[indices]

    def fit(self, X, y):
    self.X_train, self.y_train = self.bootstrap_sample(X, y)

    def predict(self, X):
    predictions = [self._predict(x) for x in X]
    return np.array(predictions)

    def _predict(self, x):
    distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
    k_indices = np.argsort(distances)[:self.k]
    k_nearest_labels = [self.y_train[i] for i in k_indices]
    most_common = Counter(k_nearest_labels).most_common(1)
    return most_common[0][0]
```

Basic Deep Learning

Q1. What are autoencoders? Explain the different layers of autoencoders and mention three practical usages of them?

Answer :

1. Autoencoders are one of the deep learning types used for unsupervised learning. There are key layers of autoencoders, which are the input layer, encoder, bottleneck hidden layer, decoder, and output.

2. The three layers of the autoencoder are:-
3. Encoder - Compresses the input data to an encoded representation which is typically much smaller than the input data.
4. Latent Space Representation/ Bottleneck/ Code - Compact summary of the input containing the most important features
5. Decoder - Decompresses the knowledge representation and reconstructs the data back from its encoded form. Then a loss function is used at the top to compare the input and output images.
   NOTE- It's a requirement that the dimensionality of the input and output be the same. Everything in the middle can be played with.

   Autoencoders have a wide variety of usage in the real world. The following are some of the popular ones:
   - Transformers and Big Bird (Autoencoders is one of these components in both algorithms): Text Summarizer, Text Generator
   - Image compression
   - Nonlinear version of PCA

Q2. What is an activation function and discuss the use of an activation function? Explain all different types of activation functions?

Answer :
In mathematical terms, the activation function serves as a gate between the current neuron input and its output, going to the next level. Basically, it decides whether neurons should be activated or not. It is used to introduce non-linearity into a model.
Activation functions are added to introduce non-linearity to the network, it doesn't matter how many layers or how many neurons your net has, the output will be linear combinations of the input in the absence of activation functions. In other words, activation functions are what make a linear regression model different from a neural network. We need non-linearity, to capture more complex features and model more complex variations that simple linear models can not capture.

There are a lot of activation functions:

- Sigmoid function: f(x) = 1/(1+exp(-x)): The output value of it is between 0 and 1, we can use it for classification. It has some problems like the gradient vanishing on the extremes, also it is computationally expensive since it uses exp.

- Tanh: f(x) = (exp(x) - exp(-x))/(exp(x)+exp(-x)): The output value of it is between -1 and 1, gives zero centered outputs with stronger gradients than sigmoid. It still has same problem of vanishing gradient but on lesser points than sigmoid functions.

- ReLU: f(x) = max(0,x): it returns 0 if the input is negative and the value of the input if the input is positive. It solves the problem of vanishing gradient for the positive side, however, the problem is still on the negative side. It is fast because we use a linear function in it.

- Leaky ReLU: F(x)= ax, x<0 F(x)= x, x>=0
  It solves the problem of vanishing gradient on both sides by returning a value “a” on the negative side and it does the same thing as ReLU for the positive side.

- Parametric ReLU: F(x) = { x if x>0, αx if x≤0}
  α in this ReLU is a learned parameter during training. It is better suited for complex problems as it can adapt α values. But it is more computationally expensive because of learning α.

- Exponential Linear Unit (ELU): f(x)={x if x>0, α(ex−1) if x≤0}
  Its value ranges from -α to infinity. It helps to mitigate vanishing gradient problems and can have negative outputs, improving the learning rate.

- Softplus: f(x)=log(1+exp(x))
  It gives positive values.
  It is similar to ReLU but differentiable everywhere.

- Softmax: f(xi) = exp(xi)/∑jexp(xi)
  it is usually used at the last layer for a classification problem because it returns a set of probabilities, where the sum of them is 1. Moreover, it is compatible with cross-entropy loss, which is usually the loss function for classification problems.

- GELU (Gaussian Error Linear Unit): f(x)=x · Φ(x) where Φ(x) is the CDF of a standard Gaussian distribution.

  It is frequently used in modern models like Transformers (e.g., BERT).

- Hard Sigmoid: f(x)=max(0,min(1,0.2x+0.5))

The output value of it is between 0 and 1, it approximates sigmoid but is computationally cheaper. It is efficient to compute, works well in resource-constrained environments.

- Swish: $f(x) = x \cdot \sigma(x)$
  It is new function used in deep learning models like those from Google (e.g., EfficientNet)
  It is smooth and non-monotonic, more computationally complex than ReLU.

Q3. What can be done if I want to have parallel trunks trained simultaneously?results?

Answer :

This can be done using parameter sharing which is the method of sharing weights by all neurons in a particular feature map. Therefore helps to reduce the number of parameters in the whole system, making it computationally cheap. It basically means that the same parameters will be used to represent different transformations in the system. This means the same matrix elements may be updated multiple times during backpropagation from varied gradients. The same set of elements will facilitate transformations at more than one layer instead of those from a single layer as conventional. In that case, using shared weights in a few layers(usually the bottom layers) helps the model converge better. This behavior, as observed, can be attributed to more diverse feature representations learned by the system. Since neurons corresponding to the same features are triggered in varied scenarios. Helps to model to generalize better.

Q4. What if all activation functions are turned to linear?

Answer :

If all activation functions in a neural network are turned to linear functions, the network will behave like a single linear model, regardless of the number of layers. This is because a composition of linear functions is still a linear function, so the network will lose its ability to model complex, non-linear relationships in the data.

Q5. If all the activation functions are positive, can output of output layers be negative?

Answer :

No, if all activation functions in the network are strictly positive, the output of the output layer cannot be negative. This is because the activations from previous layers will all be non-negative, and if the output layer only combines these positive values, the result will also be non-negative.

Q6. How to decide which activation function to use?

Answer: Choosing an activation function depends on several factors related to the problem you are trying to solve:
Non-linearity: Most deep learning problems require non-linear activation functions to model complex patterns. Functions like ReLU, sigmoid, and tanh introduce non-linearity.

Type of task:

- Classification: For binary classification, use sigmoid in the output layer. For multi-class classification, use softmax in the output layer.
- Regression: For regression tasks, use linear activation in the output layer.
- Vanishing gradient problem: To avoid vanishing gradients in deep networks, ReLU (Rectified Linear Unit) or its variants (Leaky ReLU, ELU) are commonly used because they avoid saturation in the positive range.
- Computational efficiency: ReLU is often preferred because it's simple and computationally efficient, as it does not require expensive operations like exponentiation.
- Symmetry: Functions like tanh are symmetric around zero, which can help in certain cases where negative values are important.

Q7. What is normalization in the context of deep learning, and what are the different types of normalization techniques? When would you use each of these techniques?

Answer: Normalization is a data preprocessing technique that rescales feature values to a common range, typically between 0 and 1. This process helps improve the performance and stability of machine learning algorithms by ensuring that each feature contributes equally to the distance calculations and model training.

- Batch Normalization: Normalizes the input to a layer across the batch dimension. It scales and shifts the normalized data to improve training stability and speed. Used for large batch sizes and deep CNNs (image classification, object detection).

- Layer Normalization: Normalizes across the features within a single training example rather than across the batch.Used for recurrent neural networks (NLP tasks, small batches).
- Instance Normalization: normalizes each individual training example separately across the feature channels. It was initially used in style transfer tasks,used for tasks where texture and contrast are important (style transfer).
- Group Normalization: It divides channels into groups and normalizes within each group. Unlike batch normalization, it does not rely on large batch sizes.When batch size is small (object detection, segmentation).
- Weight Normalization: This technique normalizes the weights of the neurons, decoupling the magnitude of the weights from their direction. It helps in controlling the learning dynamics. When you want to control weight magnitude (image generation).
- Spectral Normalization: Spectral normalization normalizes the weight matrices by their spectral norm (largest singular value), effectively controlling the Lipschitz constant of the neural network. Used when training GANs to stabilize and control gradient magnitudes.

Q7. What would be the implications for the mapping from input to output if the activation function in a neural network were linear (a[z]=ψ0+z*ψ1) or removed entirely (resulting in a[z]=z)?

Answer :
When activation functions become linear:
no matter how many inputs are received, with a linear activation function, the model can learn to make only linear combinations of the inputs. This basically makes the entire function linear regardless of how many layers the network has, as linearity is preserved by the property of a composition of linear functions being a linear function. The model would thus only be able to discern linear relationships in the data, severely limiting its ability to model complex patterns.
No Activation Function (Direct Mapping):
Similar to the case with a linear activation function, this setup means that the network can only represent linear mappings from input to output. Without non-linearities, no matter how deep the network is, it cannot learn complex functions or relationships, leading to the same limitation as the linear activation function.
Whether the activation function is linear or removed, the resulting mapping from input to output remains linear. This significantly limits the expressiveness of the model and its

ability to capture the underlying complexities in the data. Non-linear activation functions are essential in neural networks to allow for the modeling of more complex, non-linear relationships.

Q8. In a neural network with one input, three hidden units, and one output, how do the Heaviside step function, hyperbolic tangent function (tanh), and rectangular function (rect) affect the family of functions that the network can represent? Additionally, how do the parameters influence the output?

Answer：Heaviside Step Function:

- Outputs 1 for inputs ≥ 0 and 0 otherwise, leading to piecewise constant functions. The parameters define thresholds, creating binary classifications but limiting the model to step-like behavior.

Hyperbolic Tangent Function (tanh):

- Outputs values between -1 and 1, allowing for smooth transitions. This introduces non-linearity, enabling the model to represent continuous relationships. Parameters influence the slope and position, allowing for nuanced predictions.

Rectangular Function (rect):

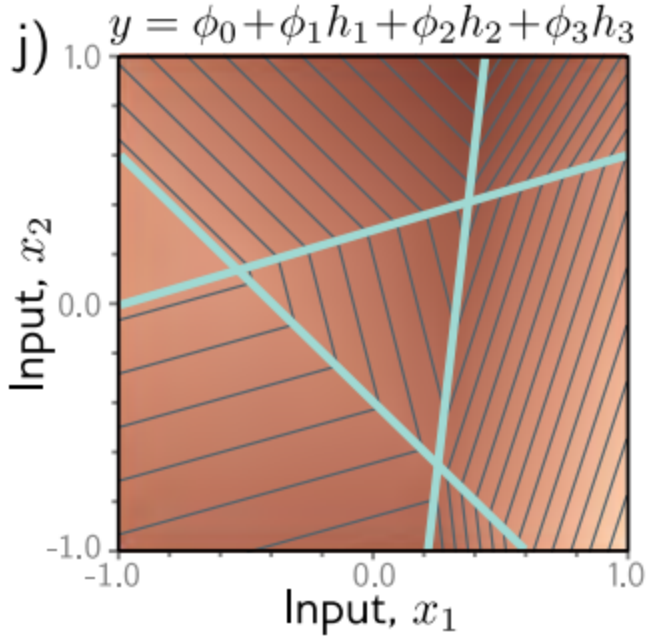
- Outputs 1 for inputs in [0, 1] and 0 otherwise, creating bounded piecewise constant outputs. The parameters define the active range, allowing for some complexity but still resulting in distinct regions.

Q9. Derive the number of regions created by the shallow network.

- For Di=2, D=3,Do = 1, calculate the maximum number of regions created by the partitioning of a 2-dimensional space with 3 hyperplanes.

Next, extend this to a case with 5 hidden units (D = 5).

- How many regions can be created in the same 2-dimensional input space if we increase the number of hidden units to 5?



Answer: To calculate the maximum number of regions created by a shallow network with Di = 2 and D = 3 hidden units, we apply Zaslavsky's formula:

$$P(D_i, D) = \sum_{j=0}^{D} \binom{D}{j}$$

Substituting the values:
Di = 2
D = 3
We have:

$$P(2,3) = \binom{3}{0} + \binom{3}{1} + \binom{3}{2}\binom{3}{3}$$

Calculating each term:

$$\binom{3}{0} = 1, \quad \binom{3}{1} = 3, \quad \binom{3}{2} = 3, \quad \binom{3}{3} = 1$$

Adding these together gives:

P(2,3)=1+3+3+1=8

However, due to the way the functions overlap and partition the space, the maximum distinct regions created in practice is seven, as shown in the figure.
For D = 5:
Now, if we increase the number of hidden units to D = 5, we calculate:

$$P(2,5) = \sum_{j=0}^{5} \binom{5}{j}$$

Calculating each term:

$$\binom{5}{0} = 1, \quad \binom{5}{1} = 5, \quad \binom{5}{2} = 10, \quad \binom{5}{3} = 10, \quad \binom{5}{4} = 5, \quad \binom{5}{5} = 1$$

Adding these together gives:

P(2,5)=1+5+10+10+5+1=32

Thus, the maximum number of regions created by a shallow network with 2-dimensional input and 5 hidden units is 32.

Q10. Consider a deep neural network with the following configuration:

- Number of inputs Di=5,
- Number of outputs Do=1,
- Number of hidden layers K=20,
- Each hidden layer contains D=30 hidden units.

Questions:

1. Define the depth of the network and calculate it based on the given information.
2. Define the width of the network and compute it based on the number of hidden units per layer.

Answer : Current number of weights:

1. Input to first hidden layer: 1×10=10
2. Hidden to hidden layers: 9×(10×10)=900
3. Last hidden to output: 10×1=10

Total: 10+900+10=920

Increasing depth by one:

Adding a hidden layer adds 10×10=100 weights.

Increasing width by one:

Each layer has 11 units now:

- Input to first hidden: 1×11=11 (adds 1 weight)
- Hidden to hidden: 9×(11×11)=1089(adds 189 weights)
- Last hidden to output: 11×1=11(adds 1 weight)
- Total increase: 1+189+1=191 weights.

Conclusion:

- Increasing depth adds 100 weights.
- Increasing width adds 191 weights.

Thus, increasing width adds more weights.

Q11. Consider a deep neural network with the following configuration:

- Number of inputs Di=5,
- Number of outputs Do=4,
- Three hidden layers with sizes D1=20, D2=10, and D3=7, respectively.

Questions:

1. Define the equations for each layer of the network, starting from the input to the output.
2. Determine the dimensions of the weight matrices and bias vectors for each layer.

Answer : First hidden layer:
$h_1 = a[\beta_1 + \Omega_1 x]$
Where:

- Ω_1 is of size 20×5 (connecting 5 inputs to 20 units in the first hidden layer).
- β_1 is of size 20×1 (bias for the 20 units).

Second hidden layer:
$h_2 = a[\beta_2 + \Omega_2 h_1]$
Where:

- Ω_2 is of size 10×20 (connecting 20 units in the first hidden layer to 10 units in the second).
- β_2 is of size 10×1 (bias for the 10 units).

Third hidden layer:
$h_3 = a[\beta_3 + \Omega_3 h_2]$
Where:

- Ω_3 is of size 7×10(connecting 10 units in the second hidden layer to 7 units in the third).
- β_3 is of size 7×1(bias for the 7 units).

Output layer:
$y = \beta_o + \Omega_o h_3$
Where:

- Ω_o is of size 4×7 (connecting 7 units in the third hidden layer to 4 output units).
- β_o is of size 4×1 (bias for the 4 outputs).

This provides the complete set of equations and the sizes of the weight matrices and bias vectors for each layer.

Q12. Consider a deep neural network with the following structure:

- A single input,
- A single output,

- K hidden layers, each containing D hidden units.

Calculate the number of parameters (weights and biases) for each layer and sum them up for the entire network.

Answer : The total number of parameters in a deep neural network with 1 input, 1 output, K hidden layers, and D hidden units per layer is derived as follows:

1. Input to first hidden layer: 2D parameters (weights + biases).
2. Hidden layers: Each of the K−1 hidden layers has D(D+1) parameters (weights + biases), contributing (K−1)D(D+1).
3. Last hidden layer to output: D+1 parameters (weights + bias).

Total parameters:

$$\text{Total}=2D+(K-1)D(D+1)+(D+1)=3D+1+(K-1)D(D+1)$$

Q12. Consider two neural networks that map a scalar input x to a scalar output y:

1. First Network (Shallow)
    - Number of hidden units D=95
2. Second Network (Deep)
    - Number of layers K=10
    - Each layer contains D=5 hidden units

Questions:

1. Determine the weights and biases for both networks.
2. Analyze the capacity for linear region generation for both the shallow and deep networks.
3. Consider the computational aspects of shallow vs. deep networks.

Answer : Using the formula from the above problem, the shallow network has 3×95+1 = 286 parameters, and the second network has 3 × 5 + 1 + (9 × 5 × 6) = 286 parameters. They are both the same. The shallow network can create 96 regions; since there is just one input, each hidden unit creates one joint, for a total of 95 joints separating 96 linear regions. The number of linear regions for the deep network is given by equation 4.17 and is 60,466,176. In principle, the shallow network will be faster to run on modern hardware as the computation is more parallel.

Regularisation

Q1. You are using a deep neural network for a prediction task. After training your model, you notice that it is strongly overfitting the training set and that the performance on the test isn't good. What can you do to reduce overfitting?

Answer : To reduce overfitting in a deep neural network changes can be made in three stages: The input data to the network, the network architecture, and the training process.

The input data to the network:

- Check if all the features are available and reliable.
- Check if the training sample distribution is the same as the validation and test set distribution. Because if there is a difference in validation set distribution then it is hard for the model to predict as these complex patterns are unknown to the model.
- Check for train / valid data contamination (or leakage)
- The dataset size is enough, if not try data augmentation to increase the data size
- Check if the dataset is balanced

Network architecture:

- Overfitting could be due to model complexity. Question each component:
- can fully connect layers be replaced with convolutional + pooling layers?
- what is the justification for the number of layers and number of neurons chosen?
- Given how hard it is to tune these, can a pre-trained model be used?
- Add regularization - lasso (l1), ridge (l2), elastic net (both)
- Add dropouts
- Add batch normalization

The training process:

Improvements in validation losses should decide when to stop training. Use callbacks for early stopping when there are no significant changes in the validation loss and restore best weights.

Q2. Why should we use Batch Normalization?

Answer : Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch.
Usually, a dataset is fed into the network in the form of batches where the distribution of the data differs for every batch size. By doing this, there might be chances of vanishing gradient or exploding gradient when it tries to backpropagate.
In order to combat these issues, we can use BN (with irreducible error) layer mostly on the inputs to the layer before the activation function in the previous layer and after fully connected layers.

Batch Normalisation has the following effects on the Neural Network:

- Robust Training of the deeper layers of the network.
- Better covariate-shift proof NN Architecture.
- Has a slight regularisation effect.
- Centred and Controlled values of Activation.
- Tries to Prevent exploding/vanishing gradient.
- Faster Training/Convergence to the minimum loss function

Q3. Why is Sigmoid or Tanh not preferred to be used as the activation function in the hidden layer of the neural network?

Answer : A common problem with Tanh or Sigmoid functions is that they saturate. Once saturated, the learning algorithms cannot adapt to the weights and enhance the performance of the model. Thus, Sigmoid or Tanh activation functions prevent the neural network from learning effectively leading to a vanishing gradient problem. The vanishing gradient problem can be addressed with the use of Rectified Linear Activation Function (ReLu) instead of sigmoid and Tanh.

Q4. How does L1/L2 regularization affect a neural network?

Answer : Overfitting occurs in more complex neural network models (many layers, many neurons) and the complexity of the neural network can be reduced by using L1 and L2 regularization as well as dropout, Data augmentation and Dropout. L1 regularization forces the weight parameters to become zero. L2 regularization forces the weight parameters towards zero (but never exactly zero|| weight decay)
Smaller weight parameters make some neurons neglectable therefore neural network becomes less complex and less overfitting.
Regularization has the following benefits:

- Reducing the variance of the model over unseen data.
- Makes it feasible to fit much more complicated models without overfitting.
- Reduces the magnitude of weights and biases.
- L1 learns sparse models that is many weights turn out to be 0.

Q5. What is the effect of dropout on the training and prediction speed of your deep learning model?

Answer: Dropout is a regularization technique, which zeroes down some weights and scales up the rest of the weights by a factor of 1/(1-p). Let's say if Dropout layer is initialized with p=0.5, that means half of the weights will zeroed down, and rest will be scaled by a factor of 2. This layer is only enabled during training and is disabled during validation and testing. Hence validation and testing is faster. The reason why it works only during training is, we want to reduce the complexity of the model so that model doesn't overfit. Once the model is trained, it doesn't make sense to keep that layer enabled.

Q6. What is the advantage of deep learning over traditional machine learning?

Answer: Deep learning offers several advantages over traditional machine learning approaches, including:

- Ability to process large amounts of data: Deep learning models can analyze and process massive amounts of data quickly and accurately, making it ideal for tasks such as image recognition or natural language processing.

- Automated feature extraction: In traditional machine learning, feature engineering is a crucial step in the model building process. Deep learning models, on the other hand, can automatically learn and extract features from the raw data, reducing the need for human intervention.
- Better accuracy: Deep learning models have shown to achieve higher accuracy levels in complex tasks such as speech recognition and image classification when compared to traditional machine learning models.
- Adaptability to new data: Deep learning models can adapt and learn from new data, making them suitable for use in dynamic and ever-changing environments.
- While deep learning does have its advantages, it also has some limitations, such as requiring large amounts of data and computational resources, making it unsuitable for some applications.

Q7. For a given small dataset what should be used deep learning or machine learning? If you choose to use a neural network, what strategies can be employed to enhance its performance?

Answer: When working with a small dataset, traditional machine learning methods are often preferred because they require fewer data points to train effectively and are less prone to overfitting compared to deep learning models.
However, if you want to use a neural network for such a dataset, you can employ several strategies to improve its performance:

- Data Augmentation: This technique involves artificially increasing the size of your dataset by applying transformations such as rotation, scaling, and flipping to the existing data, which helps the model generalize better.
- Transfer Learning: You can leverage pre-trained models that have been trained on larger datasets. Fine-tuning these models on your small dataset can lead to better performance without requiring a large amount of data.
- Regularization Techniques: Implement techniques such as dropout or L2 regularization to prevent overfitting, which is a common issue when using neural networks on small datasets.
- Simplifying the Model: Use a simpler architecture with fewer layers and neurons to reduce the risk of overfitting.
- Padding: If your input data varies in size (like images), use padding to ensure consistent input dimensions. Padding adds extra pixels (zeros or specific values) around the borders of the input data, allowing the neural network to process inputs of uniform size.

By using these strategies, you can effectively utilize a neural network even with a small dataset.

Q8. Difference between SGD and Adam and which is better in which situation?

Answer: SGD with momentum can find lower minima faster than Adam, which generalizes better over a variety of deep learning tasks. However, this is strange since SGD is a special case of Adam (when β = 0, γ = 1) once the modification term in becomes one, which happens quickly.

$$\tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1-\beta^{t+1}}$$
$$\tilde{\mathbf{v}}_{t+1} \leftarrow \frac{\mathbf{v}_{t+1}}{1-\gamma^{t+1}}.$$

where β and γ are the momentum coefficients for the two statistics and mt+1 is the gradient and vt+1 is the pointwise squared gradient.

It is hence more likely that SGD outperforms Adam when we use Adam's default hyperparameters. AdamW substantially improves the performance of Adam in the presence of L2 regularization. If we search for the best Adam hyperparameters, it performs just as well as SGD and converges faster. There is a method called SWATS that starts using Adam (to make rapid initial progress) and then switches to SGD (to get better final generalization performance).

Q9.What causes exploding gradients? Is it caused by an optimiser or regularizer?

Answer: Exploding gradients are primarily caused by deep networks and poor initialization of weights, where large gradients get propagated back through the layers during training. This can result from:

- Very deep architectures (e.g., RNNs and deep feedforward networks).
- Poor weight initialization.
- High learning rates.

Optimizers: Optimizers themselves do not directly cause exploding gradients, but they can exacerbate the problem. For example, if the learning rate is too high in optimizers like SGD, it can make the weight updates too large, worsening the effect of exploding gradients.

Regularizers: Regularization techniques (e.g., L2 regularization) generally help mitigate overfitting, but they are not the cause of exploding gradients. They can sometimes help by limiting weight growth, which indirectly reduces the impact of exploding gradients.

Solutions of exploding gradient:
Gradient clipping: Limits the size of the gradients to prevent them from growing too large.
Proper weight initialization: Techniques like Xavier or He initialization can help mitigate the problem.
Use of specialized optimizers: Adam and RMSProp handle adaptive learning rates, reducing the risk of exploding gradients.
For a layer with nin input units and nout output units, the weights are initialized using the following formula:
W~N(0,2/(nin+nout))

Q10. If you have a deep learning model that expects an input layer with 100 neurons, but your dataset only has 51 features (neurons), what strategies can you employ to address this discrepancy?

Answer:
Padding: We can pad the input features to meet the required number of neurons. This involves adding additional neurons with a fixed value (often zeros) to increase the input dimension from 51 to 100. While this is a straightforward solution, it's essential to ensure that the padded values do not negatively impact model performance.

PCA: we can consider dimensionality reduction techniques like PCA (Principal Component Analysis) to transform the original feature space. Although PCA reduces dimensions, you can expand it back to 100 dimensions by using methods like interpolation or adding synthetic features to fill the gap.

Feature engineering techniques: used to create new features that might enrich the dataset. This could involve creating interaction terms, polynomial features, or using domain knowledge to extract meaningful information from existing features.

Q11. What are the limitations in deep learning?

Answer: As models reach state-of-the-art performance, further improvements can become increasingly challenging. Achieving just a 0.01% increase in accuracy often requires substantial effort, resources, and experimentation, making it less practical for many applications.
Deep learning models can be sensitive to adversarial examples, where small, intentionally crafted perturbations in input data can lead to significant misclassifications.
Deep learning systems can perpetuate biases present in training data, leading to unfair or discriminatory outcomes. This raises ethical concerns about their deployment in real-world applications.

Q12. Imagine you're building a model to predict how many pedestrians will pass a specific point in the city over the next minute. You have data that includes factors like the time of day, the location (longitude and latitude), and the type of neighborhood. Since pedestrian counts can be modeled well using the Poisson distribution, which has a single positive parameter λ\lambdaλ representing the average rate of pedestrians, how would you go about designing a loss function for your model? Assume you have III training pairs of data, {(xi,yi)}.

$$Pr(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}.$$

Design a loss function for this model assuming we have access to I training pairs {xi, yi}.

Answer: To predict the number of pedestrians using the Poisson distribution, we can minimize the negative log-likelihood. The Poisson probability mass function is:

$$P(y = k \mid \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

The log-likelihood for a single training example (xi,yi) is:

$$\log P(y_i \mid \lambda_i) = y_i \log(\lambda_i) - \lambda_i$$

The negative log-likelihood loss function over all training examples is:

$$L(\lambda) = -\sum_{i=1}^{I} (y_i \log(\lambda_i) - \lambda_i)$$

Here, $\lambda_i = f(x_i)$ is the model's predicted rate, and yi is the observed count. This loss can be used to train the model.

Optimization

Q1. How to know whether your model is suffering from the problem of Exploding Gradients?

Answer: By taking incremental steps towards the minimal value, the gradient descent algorithm aims to minimize the error. The weights and biases in a neural network are updated using these processes. However, at times, the steps grow excessively large, resulting in increased updates to weights and bias terms to the point where the weights overflow (or become NaN, that is, Not a Number). An exploding gradient is the result of this, and it is an unstable method.
There are some subtle signs that you may be suffering from exploding gradients during the training of your network, such as:

- The model is unable to get traction on your training data (e g. poor loss).
- The model is unstable, resulting in large changes in loss from update to update.
- The model loss goes to NaN during training.

If you have these types of problems, you can dig deeper to see if you have a problem with exploding gradients. There are some less subtle signs that you can use to confirm that you have exploding gradients:

- The model weights quickly become very large during training.
- The model weights go to NaN values during training.
- The error gradient values are consistently above 1.0 for each node and layer during training.

Q2. What is the Vanishing Gradient Problem in Artificial Neural Networks and How to fix it?

Answer: The vanishing gradient problem is encountered in artificial neural networks with gradient-based learning methods and backpropagation. In these learning methods, each of the weights of the neural network receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. Sometimes when gradients become vanishingly small, this prevents the weight to change value.

When the neural network has many hidden layers, the gradients in the earlier layers will become very low as we multiply the derivatives of each layer. As a result, learning in the earlier layers becomes very slow which can be used in learning. This problem of vanishing gradient descent happens when training neural networks with many layers because the gradient diminishes dramatically as it propagates backward through the network.

Some ways to fix it are:

- Use skip/residual connections.
- Using ReLU or Leaky ReLU over sigmoid and tanh activation functions.
- Use models that help propagate gradients to earlier time steps like in GRUs and LSTMs.

Q3.When it comes to training an artificial neural network, what could be the reason why the loss doesn't decrease in a few epochs? What can be done for this?

Answer: Some of the reasons why the loss doesn't decrease after a few Epochs are:
a) The model is under-fitting the training data.
b) The learning rate of the model is large.
c) The initialization is not proper (like all the weights initialized with 0 doesn't make the network learn any function)
d) The Regularisation hyper-parameter is quite large.
e). The classic case of vanishing gradients

Q4.Explain how backpropagation works in a neural network? Please give code for this backpropagation and also provide dry run.

Answer: Backpropagation is the algorithm used to train neural networks by minimizing the error between the predicted output and the actual output. It consists of two main phases: the forward pass, where predictions are made, and the backward pass, where gradients are computed and weights are updated based on the error.

```
def backward(self, X, y, output, learning_rate):
   output_error = y - output
   output_delta = output_error * sigmoid_derivative(output)
   self.weights_hidden_output += np.dot(self.hidden_layer_output.T,
output_delta)*
                                 learning_rate
```

```
self.bias_output += np.sum(output_delta, axis=0) * learning_rate
hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
hidden_delta = hidden_error * sigmoid_derivative(self.hidden_layer_output)
self.weights_input_hidden += np.dot(X.T, hidden_delta) * learning_rate
self.bias_hidden += np.sum(hidden_delta, axis=0) * learning_rate
```

Q5. What is optimiser and state type of optimisers and when to use which type of optimiser?

Answer: An optimizer in deep learning is an algorithm used to adjust the weights and biases of a neural network during training, in order to minimize the loss function. Optimizers guide the model in finding the optimal set of parameters (weights) to improve its predictive accuracy. They do this by updating the weights in response to the gradients computed during backpropagation.

Types of Optimiser:

1. Stochastic Gradient Descent (SGD): A variant of gradient descent that updates weights for each training sample rather than the entire dataset. Best for convex loss functions and when the dataset is very large. It works well with shallow networks or when training stability is not an issue.One problem is that it can converge slowly and may struggle with complex, non-convex loss surfaces.
2. Momentum-based SGD: Extends SGD by adding a momentum term to speed up learning and avoid getting stuck in local minima. Helps accelerate learning in deep networks and when facing high-gradient noise. Use it when you want faster convergence than plain SGD.
3. Nesterov Accelerated Gradient (NAG): A variation of momentum-based SGD that looks ahead by adjusting the gradient before the parameter update. Used when training deep neural networks and you want an even faster convergence with a more refined approach to momentum.
4. Adagrad (Adaptive Gradient Algorithm): An optimizer that adapts the learning rate for each parameter based on the history of gradients. Effective for sparse data and features (e.g., NLP tasks). It adapts well to infrequent features but suffers from decaying learning rates over time.
5. RMSProp (Root Mean Square Propagation): An optimizer that normalizes the gradients using a moving average of squared gradients. It works well in online settings, recurrent

neural networks, and for non-stationary objectives. Best for tasks where Adagrad would slow down due to diminishing learning rates.

6. Adam (Adaptive Moment Estimation): Combines the advantages of both momentum-based SGD and RMSProp by using moving averages of both the gradient and its squared values. Widely used in a variety of tasks and architectures, especially for deep learning models. It's highly effective for large-scale problems and sparse data.
7. AdaMax: A variant of Adam that uses the infinity norm of the gradient for weight updates. It is useful when the Adam optimizer fails due to exploding gradients or when Adam's assumptions do not hold well.
8. Nadam: A combination of Adam and Nesterov accelerated gradient, adding a momentum term to Adam.It is useful for tasks where both Adam and NAG are suitable, offering faster convergence.

Q6 . What is regularization and state type of optimisers and when to use which type?

Answer:

Regularization is a technique in machine learning and deep learning used to prevent overfitting by penalizing or constraining the model's complexity. Overfitting occurs when a model learns the noise in the training data, performing well on it but poorly on unseen data (test data). Regularization helps the model generalize better by discouraging overly complex models that fit the training data too closely.

Types of Regularization:

- L1 Regularization: Adds a penalty equal to the absolute value of the weights in the loss function. Used for feature selection, sparse models, or when you believe some features are irrelevant.
- L2 Regularization: Adds a penalty proportional to the square of the weights in the loss function. Used for models where you want to reduce overfitting without eliminating any features (common in dense feature spaces).
- Elastic Net: A combination of L1 and L2 regularization. Used for balancing between L1 and L2, especially for high-dimensional data.
- Dropout: A technique where randomly selected neurons are ignored during each forward and backward pass forcing network not to rely too much on any particular neuron. It is used for deep networks, especially convolutional and recurrent neural networks, to prevent overfitting.
- Early Stopping: A technique where training is stopped as soon as the model's performance on validation data starts to degrade (to avoid overfitting). For training

with a limited number of epochs to prevent overfitting based on validation performance.

- Max-Norm Regularization: Constrains the weights to not exceed a fixed maximum norm. To constrain the weights and prevent the network from relying too heavily on any particular neuron.

Q7. What is the difference between odds and probability? How are each defined? How are log(odds) related to logistic regression?

Answer: Odds are the ratio of something happening to that something not happening, whereas probability is the ratio of something happening to the total number of possibilities.

The probability in logistic regression is modeled as $$P(X) = \frac{e^{\beta 0+\beta 1*x}}{1+e^{\beta 0+\beta 1*x}}$$

Q8. You're working on a multivariate regression model to predict a person's height (in meters) and weight (in kilos) based on some input data x. Given that height and weight have very different ranges, what potential problems could arise from this disparity? Additionally, propose two solutions to address these issues effectively?

Answer:
Problems:
1. Large Scales: It may make the height vs. weight ~50-100+kg about determining the model's focus to better predict the weight over height with biased predictions.

2. Slow Optimization: Large differences in magnitudes may reduce optimization where the model might not be optimized to learn it faster.

Solutions:
1. Feature Scaling: Standardize or normalize both height and weight by ensuring that they are on the same scale, for example through subtracting the mean and then dividing by the standard deviation.

2. Weighted Loss Function: Use a weighted loss function whereby different weights are assigned to different target variables, depending on their scale. This model is able to treat the output of each target equally and usually yields better results during training.

Q9. Can (non-stochastic) gradient descent with a fixed learning rate escape local minima?

Answer:

Yes! The distance moved just depends on the gradient at the current point and the learning rate. The movement pays no attention to whether it crosses from valley to valley. Usually, the learning rate is very small though, so this may not actually happen in practice.

Hyperparameter & Tuning

Q1.Name and explain a few hyperparameters used for training a neural network?

Answer:

Hyperparameters are any parameter in the model that affects the performance but is not learned from the data unlike parameters (weights and biases), the only way to change it is manually by the user.

- Number of nodes: number of inputs in each layer.
- Batch normalization: normalization of inputs in a layer.
- Learning rate: the rate at which weights are updated.
- Dropout rate: percent of nodes to drop temporarily during the forward pass.
- Kernel: matrix to perform dot product of image array with
- Activation function: defines how the weighted sum of inputs is transformed into outputs (e.g. tanh, sigmoid, softmax, Relu, etc)
- Number of epochs: number of passes an algorithm has to perform for training
- Batch size: number of samples to pass through the algorithm individually. E.g. if the dataset has 1000 records and we set a batch size of 100 then the dataset will be divided into 10 batches which will be propagated to the algorithm one after another.

- Momentum: Momentum can be seen as a learning rate adaptation technique that adds a fraction of the past update vector to the current update vector. This helps damps oscillations and speed up progress towards the minimum.
- Optimizers: They focus on getting the learning rate right.
- Adagrad optimizer: Adagrad uses a large learning rate for infrequent features and a smaller learning rate for frequent features.
- Other optimizers, like Adadelta, RMSProp, and Adam, make further improvements to fine-tuning the learning rate and momentum to get to the optimal weights and bias. Thus getting the learning rate right is key to well-trained models.
- Learning Rate: Controls how much to update weights & bias (w+b) terms after training on each batch. Several helpers are used to getting the learning rate right.

Q2.What might happen if you set the momentum hyperparameter too close to 1 (e.g., 0.9999) when using an SGD optimizer?

Answer:

If the momentum hyperparameter is set too close to 1 (e.g., 0.99999) when using an SGD optimizer, then the algorithm will likely pick up a lot of speed, hopefully moving roughly toward the global minimum, but its momentum will carry it right past the minimum.
Then it will slow down and come back, accelerate again, overshoot again, and so on. It may oscillate this way many times before converging, so overall it will take much longer to converge than with a smaller momentum value.
Also since the momentum is used to update the weights based on an "exponential moving average" of all the previous gradients instead of the current gradient only, this in some sense, combats the instability of the gradients that comes with stochastic gradient descent, the higher the momentum term, the stronger the influence of previous gradients to the current optimization step (with the more recent gradients having even stronger influence), setting a momentum term close to 1, will result in a gradient that is almost a sum of all the previous gradients basically, which might result in an exploding gradient scenario.

Q3. What are the hyperparameters that can be optimized for the batch normalization layer?

Answer:

The γ and β hyperparameters for the batch normalization layer are learned end to end by the network. In batch-normalization, the outputs of the intermediate layers are normalized to have a mean of 0 and standard deviation of 1. Rescaling by γ and shifting by β helps us change the mean and standard deviation to other values.

Q4. What are the hyperparameters that can be optimized for the batch normalization layer?

Answer: The γ and β hyperparameters for the batch normalization layer are learned end to end by the network. In batch-normalization, the outputs of the intermediate layers are normalized to have a mean of 0 and standard deviation of 1. Rescaling by γ and shifting by β helps us change the mean and standard deviation to other values.

Q5. How do you choose hyperparameters for deep learning models?

Answer: Choosing hyperparameters for deep learning models is not that easy because of the complexity and size of neural networks.
The following are the main strategies:
Understanding the Model Architecture

- Learning Rate: Methods such as learning rate scheduling or adaptive learning rates are commonly employed; such methods are applied inside optimizers like Adam or RMSprop.
- Grid Search or Random Search: This method checks a predefined list of hyperparameter values.
- Bayesian Optimization: efficiently searches the hyperparameter space. This is achieved by developing a probabilistic model that predicts performance based on past evaluations.
- Cross Validation: the robustness of hyperparameter choices could be assured. This way, selected hyperparameters lead to models generalizing well for unseen data.
- Early Stopping: This may prevent overfitting and will also alert you if you need to adjust hyperparameters, such as a batch size or number of epochs.
- Hyperparameter Tuning Frameworks: The advantage of using automated hyperparameter tuning frameworks is the ease with which they help to execute the tuning process and present indications based on the past performance.
- Iterative Approach: Hyperparameter tuning is typically an iterative process. Start with broad ranges and narrow them down according to the performance of your model, learned from previous experimental results.

- Batch Size and Regularization: Experimenting with batch size and regularization as this is a significant factor in getting models to converge and generalize well.

With such strategies and methodology in your approach, you can adequately settle on the choice of hyperparameters that improve performance and robustness in deep learning models.

Q6. Give us an example of using transfer learning to fine-tune a pre-trained deep learning model for a new task?

Answer: VGG, BERT, or ResNet, are well-known examples of pre-trained models that can be loaded for transfer learning and fine-tuning purposes. Specifically, the process involves replacing the model head, i.e. the final classification layer, with a new one suited to the target task.
After this slight structural change to the model architecture, we proceed to retrain it on a new dataset using a low learning rate to adapt the model weights to the new task while major features originally learned by the pre-trained models are mostly kept.

Computer Vision

Q1. Explain the concept of image segmentation and its applications.

Answer:

- Image segmentation is a computer vision technique that partitions a digital image into discrete groups of pixels — image segments — to inform object detection and related tasks. By parsing an image's complex visual data into specifically shaped segments, image segmentation enables faster, more advanced image processing.
- Conventional image segmentation algorithms process high-level visual features of each pixel, like color or brightness, to identify object boundaries and background regions.
- Being a highly versatile and practical method of computer vision, image segmentation has a wide variety of artificial intelligence use cases, from aiding diagnosis in medical imaging to automating locomotion for robotics and self-driving cars to identifying objects of interest in satellite images.

Q2. What is object detection, and how does it differ from image classification?

Answer:

- Object detection is a computAnswer:
- Image segmentation is a computer vision technique that partitions a digital image into discrete groups of pixels — image segments — to inform object detection and related tasks. By parsing an image's complex visual data into specifically shaped segments, image segmentation enables faster, more advanced image processing.
- Conventional image segmentation algorithms process high-level visual features of each pixel, like color or brightness, to identify object boundaries and background regions.
- Being a highly versatile and practical method of computer vision, image segmentation has a wide variety of artificial intelligence use cases, from aiding diagnosis in medical imaging to automating locomotion for robotics and self-driving cars to identifying objects of interest in satellite images.
- er vision task that aims to locate objects in digital images. As such, it is an instance of artificial intelligence that consists of training computers to see as humans do, specifically by recognizing and classifying objects according to semantic categories.
- Object localization is a technique for determining the location of specific objects in an image by demarcating the object through a bounding box.
- Object classification is another technique that determines to which category a detected object belongs.
- The object detection task combines subtasks of object localization and classification to simultaneously estimate the location and type of object instances in one or more images.
- Image segmentation (or semantic segmentation) is similar to object detection, albeit more precise. Like object detection, segmentation delineates objects in an image according to semantic categories. But rather than mark objects using boxes, segmentation demarcates objects at the pixel level.

Q3. Explain how YOLO's "one-stage" detection framework differs from the "two-stage" approach of Faster R-CNN. What are the trade-offs in terms of speed and accuracy?

Answer: YOLO (One-Stage Detection)

- YOLO is a one-stage object detection framework that performs both localization and classification in a single pass through the neural network.

- Faster inference time: YOLO's single-pass design makes it computationally efficient and well-suited for real-time applications.
- Simplified architecture: The lack of a separate region proposal step simplifies the model and reduces complexity.
- Lower accuracy: The one-stage approach can be less accurate than two-stage detectors, especially for smaller objects.

Faster R-CNN (Two-Stage Detection)

- Faster R-CNN is a two-stage object detection framework that first generates region proposals and then classifies and refines those proposals.
  - Higher accuracy: The separate stages allow the model to focus on region proposals and classification/regression independently, leading to better performance.
  - Flexibility: The two-stage design provides more flexibility in terms of customizing and optimizing each component.
  - Slower inference time: The additional region proposal step makes Faster R-CNN computationally more expensive than one-stage detectors.

Trade-offs and Considerations When choosing between one-stage and two-stage object detection frameworks, the primary trade-off is between speed and accuracy. YOLO's one-stage approach is generally faster, while Faster R-CNN's two-stage design tends to be more accurate, especially for small objects.

Q3. SSD (Single Shot Detector) relies on different scale feature maps for object detection. How does this architecture help with detecting objects of varying sizes?

Answer:

1. Scale Matters for Object Detection:
   - Perspective changes the scale of objects in an image. Larger objects appear bigger, while smaller objects appear smaller.
2. Using Lower-Resolution Feature Maps for Larger Objects:
   - As the CNN processes the image, the spatial dimensions of the feature maps decrease.
   - SSD uses these lower-resolution feature maps (e.g. 4x4) to detect larger-scale objects in the image.
3. Using Higher Resolution Feature Maps for Smaller Objects:

    - The higher resolution feature maps (e.g. 38x38) from earlier CNN layers are used to detect smaller objects.
4. Customized Default Bounding Boxes per Feature Map Layer:
    - SSD defines a set of default bounding boxes for each feature map layer, with the scale of the boxes increasing from the earlier to later layers.
    - This allows the model to have the appropriate default box sizes to match the expected object sizes at each resolution.

By leveraging multi-scale feature maps and customized default boxes, SSD is able to effectively detect objects of varying sizes within a single pass through the network.

Q4. Explain the Scale-Invariant Feature Transform (SIFT) algorithm.

Answer: Scale-invariant feature Transform (SIFT) is a widely used method for keypoint detection and local feature extraction. It's robust to changes like scale, rotation, and illumination, making it effective for tasks such as object recognition, image stitching, and 3D reconstruction.

SIFT vs. Modern Techniques
While SIFT has long been a staple in the computer vision community, recent years have introduced alternative methods. For instance, convolutional neural networks (CNNs) are increasingly being used to learn discriminative image features, especially due to their effectiveness in large-scale, real-world applications.

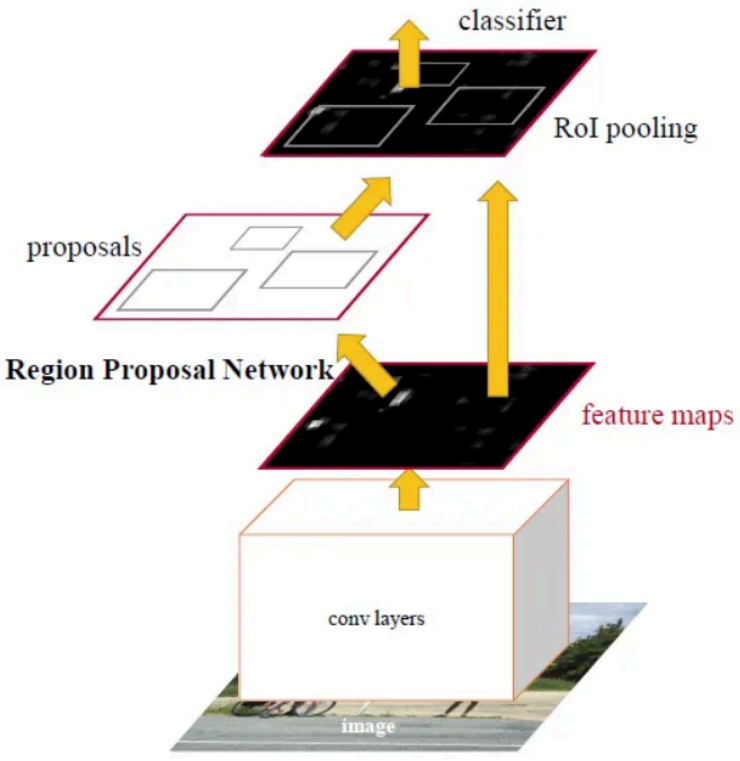
Code Example: SIFT Using OpenCV
Here is the Python code:

```
# Import the required libraries
import cv2

# Load the image in grayscale
image = cv2.imread('image.jpg', 0)

# Create an SIFT object
sift = cv2.SIFT_create()

# Detect keypoints and descriptors
keypoints, descriptors = sift.detectAndCompute(image, None)
```

```
# Visualize the keypoints
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)
cv2.imshow('Image with Keypoints', image_with_keypoints)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Q5. Faster R-CNN introduces the Region Proposal Network (RPN). Can you explain how RPN improves upon traditional region proposal methods and where is it inserted in the R-CNN architecture?

Answer: Region Proposal Network (RPN), which is, as its name suggests, used to generate object proposals. So the primary differentiator for Faster R-CNN is the RPN which is inserted after the last convolutional layer.
This is trained to produce region proposals directly without the need for any external mechanism like Selective Search. After this we use ROI pooling and an upstream classifier and bounding box regressor similar to Fast R-CNN.



Q6. How does YOLO achieve real-time object detection, and what optimizations are responsible for this speed?

Answer: Key optimizations:

1. One-Stage vs. Two-Stage Approach:
    - YOLO uses a one-stage object detection approach, as opposed to the two-stage approach of detectors like Faster R-CNN.
2. Predicting Bounding Boxes and Classes Simultaneously:
    - For each grid cell, YOLO predicts the bounding boxes, objectness scores, and class probabilities all at once.
    - This single forward pass through the network is much faster than the sequential region proposal and classification steps in a two-stage detector.
3. Smaller and Simpler Network Architecture:
    - The simpler architecture means fewer parameters and computations, leading to faster inference times.
4. Non-Maximum Suppression (NMS) Optimization:
    - YOLO applies NMS to remove duplicate detections efficiently, further improving the speed of the inference process.
5. Batch Processing:
    - YOLO can process multiple images in a batch, leveraging the parallelization capabilities of modern hardware (e.g., GPUs) to achieve even faster inference times.

The combination of the one-stage design, simplified architecture, resolution tradeoffs, and efficient post-processing techniques allows YOLO to achieve impressive real-time object detection speeds, often reaching 30-60 frames per second (FPS) or more, depending on the specific hardware and model version used.

Q7. What is the role of Feature Pyramid Networks (FPN) in object detection? How does it enable multi-scale detection?

Answer:

1. Capturing Multi-scale Features:
    - FPN leverages the hierarchical structure or backbone of a convolutional neural network (CNN) to capture features at multiple scales.
2. Feature Fusion and Upsampling:

    - FPN performs feature fusion by combining the feature maps from different levels of the pyramid.
    - It uses lateral connections to fuse the semantically strong but spatially coarse features from the higher levels with the spatially fine but semantically weak features from the lower levels.
3. Multi-scale Object Detection:
    - The resulting feature pyramid contains rich multi-scale features, which are then used by the object detection head (e.g., region proposal network, classification, and bounding box regression).
    - Different levels of the feature pyramid are used to detect objects of varying sizes.
    - The higher levels of the pyramid are responsible for detecting larger objects, while the lower levels handle the detection of smaller objects.

The key advantage of FPN is that it allows the object detection model to effectively handle objects of different scales within a single, unified framework. By leveraging the inherent multi-scale nature of CNN feature maps and combining them through the feature pyramid, FPN enables efficient and accurate detection of objects at various scales.

Q8. How would you optimize the computational cost of a Feature Pyramid Network (FPN) without significantly compromising accuracy?

Answer:

1. Pyramid Levels Optimization:
    - Analyze the contribution of each pyramid level to the overall detection performance and reduce the number of pyramid levels or adjust their spatial dimensions to reduce the computational cost.
2. Efficient Backbone Network:
    - Choose a more lightweight and efficient backbone network (e.g., MobileNet, EfficientNet) instead of a large, complex model like ResNet.
3. Feature Map Compression:
    - Investigate techniques to compress the feature maps within the FPN, such as:
        - Channel pruning
        - Low-rank factorization
        - Quantization
4. Spatial Resolution Reduction:
    - Reduce the spatial resolution of the feature maps in the FPN, especially at the higher pyramid levels.

- This can be achieved by using strided convolutions or dilated convolutions instead of regular convolutions.

The specific optimization techniques to apply will depend on the target hardware, the required performance characteristics, and the trade-offs between accuracy and computational cost. It's essential to evaluate the impact of these optimizations on the overall detection performance and choose the most suitable approach for your application.

Q9. In the SSD (Single Shot Multibox Detector), each feature map layer is responsible for detecting objects at different scales. Suppose that for a specific feature map layer, the default boundary box scale is 0.4, and the target aspect ratios are 1, 2, 3, 1/2, and 1/3.

A. Calculate the width and height of the default boundary boxes for the given feature map layer with the following target aspect ratios:
B. If the number of cells in this feature map layer is 16×16, how many default boundary boxes will be generated for this layer?

Answer:
Part A:

Given $s = 0.4$, for each target aspect ratio r:

1. For $r = 1$:
   $w = s = 0.4, h = s = 0.4$
2. For $r = 2$:
   $w = s \times \sqrt{2} = 0.4 \times \sqrt{2} \approx 0.4 \times 1.414 \approx 0.566$
   $h = s \div \sqrt{2} = 0.4 \div 1.414 \approx 0.283$
3. For $r = 3$:
   $w = s \times \sqrt{3} = 0.4 \times \sqrt{3} \approx 0.4 \times 1.732 \approx 0.693$
   $h = s \div \sqrt{3} = 0.4 \div 1.732 \approx 0.231$
4. For $r = 1/2$:
   $w = s \times \sqrt{1/2} = 0.4 \times \sqrt{0.5} \approx 0.4 \times 0.707 \approx 0.283$
   $h = s \div \sqrt{1/2} = 0.4 \div 0.707 \approx 0.566$
5. For $r = 1/3$:
   $w = s \times \sqrt{1/3} = 0.4 \times \sqrt{0.333} \approx 0.4 \times 0.577 \approx 0.231$
   $h = s \div \sqrt{1/3} = 0.4 \div 0.577 \approx 0.693$

Part B:

For each cell in the 1616×16 feature map, there are 6 default boundary boxes (5 aspect ratios plus 1 additional scale).
Thus, the total number of default boundary boxes = 16×16×6==1536 boundary boxes.

Q10. In an object detection task, you are using the SSD (Single Shot Multibox Detector) model, which applies Non-Maximum Suppression (NMS) to eliminate duplicate predictions of the same object. SSD sorts the predictions by confidence score, starting with the highest. For a

given prediction, SSD compares it with previously considered bounding boxes of the same class and removes any predictions that have an Intersection over Union (IoU) greater than 0.45.

Now, consider the following scenario:

You have detected multiple bounding boxes for objects in an image, with confidence scores and IoU values provided below. After sorting by confidence score, the following predictions are evaluated for a given class:

- Box A: Confidence = 0.95, IoU with Box B = 0.65, IoU with Box C = 0.30
- Box B: Confidence = 0.90, IoU with Box C = 0.50
- Box C: Confidence = 0.85

Task 1:
Explain which bounding boxes will be kept after applying Non-Maximum Suppression with an IoU threshold of 0.45.

Task 2:
Describe the steps followed by SSD's NMS algorithm to arrive at this result.

Answer:

Task 1:

- Box A has the highest confidence score (0.95) and is retained.
- Box B has an IoU of 0.65 with Box A, which is higher than the threshold of 0.45. Therefore, Box B will be removed because it likely overlaps significantly with Box A.
- Box C has an IoU of 0.30 with Box A (below the threshold of 0.45), so it will be retained.

Final result:

- Retained Boxes: A and C

- Removed Box: B

Task 2:

1. Step 1: Sort predictions by confidence score.
    - In this case: Box A (0.95), Box B (0.90), Box C (0.85).
2. Step 2: Start with the box having the highest confidence score (Box A).
    - Retain Box A since it has the highest score.
3. Step 3: Compare Box B with Box A using IoU.
    - IoU(Box A, Box B) = 0.65, which is greater than the threshold of 0.45.
    - Remove Box B because it overlaps significantly with Box A.
4. Step 4: Compare Box C with Box A using IoU.
    - IoU(Box A, Box C) = 0.30, which is less than 0.45.
    - Retain Box C since the overlap is small.

Conclusion: After applying NMS, only Box A and Box C are kept, while Box B is removed due to significant overlap with Box A.

Q11: In computer vision, tasks like object detection, semantic segmentation, and instance segmentation are commonly used for different applications.

Consider the following scenarios:

1. Scenario 1: You are building a deep learning model that needs to detect and draw bounding boxes around objects of interest in an image (e.g., identifying multiple cars in a traffic scene).
    - Question 1a: Which type of task is being performed here, and how does the model decide where to place the bounding boxes?
    - Question 1b: What would be the neural network architecture's final output in this case?
2. Scenario 2: In a different application, you want the model to classify each pixel in an image to determine which parts belong

to different object classes, without distinguishing between different instances of the same class (e.g., identifying all pixels that belong to the road, sky, or cars in the image).
    - Question 2a: Which task are you performing now, and what is the neural network trained to output?
    - Question 2b: How does this task differ from instance segmentation?
3. Scenario 3: You want your model to go one step further and identify individual instances of objects (e.g., distinguishing between different cars in an image).
    - Question 3a: Which task is being performed now, and how is it different from semantic segmentation?
    - Question 3b: How does the Mask R-CNN architecture solve this problem, and what are the two main components of the Mask R-CNN model?
    - Question 3c: Explain how RoI Align and binary mask classification are used to refine the outputs in this model.

Answer:

part 1:

- Question 1a: This is an object detection task, where the model detects and draws bounding boxes around objects in the image.
- Question 1b: The model outputs the coordinates of bounding boxes (x, y, width, height) along with class probabilities for each detected object.

part 2:

- Question 2a: This is a semantic segmentation task, where the model classifies each pixel into object classes like road, sky, or cars.

- Question 2b: It differs from instance segmentation because semantic segmentation does not distinguish between different instances of the same class (e.g., all cars are labeled the same).

part 3:

- Question 3a: This is instance segmentation, where the model identifies and segments each individual object instance (e.g., different cars).
- Question 3b: Mask R-CNN solves this by first detecting bounding boxes (object detection) and then performing segmentation on each box.
- Question 3c: RoI Align refines bounding boxes, and the binary mask classifier performs segmentation within each box by distinguishing between the object and the background (1/0).

Q12: Consider that you have 3 cows in an image, and you apply Semantic Segmentation to it.
Will the model be able to distinguish between the 3 cows, or will it just identify the area they occupy?

Answer: In semantic segmentation, the model classifies each pixel in an image to a specific class (e.g., cow, background). This means that while it can identify the presence of cows and distinguish the areas they occupy, it does not differentiate between individual cows. All cows in the image would be assigned the same label, typically represented by a single color for that class.

If you need to distinguish between individual cows (e.g., to count them separately), you would use instance segmentation instead. Instance segmentation not only classifies each pixel but also distinguishes between different instances of the same class (e.g., Cow 1, Cow 2, Cow 3), allowing for unique identification of each cow in the image.

Q13. Explain how attention weights are computed in a typical attention mechanism. Illustrate the process with a mathematical example.

Answer:

In an attention mechanism, the goal is to compute a weighted representation of a set of inputs based on their relevance to a specific task or output. The basic idea is to assign different weights to different parts of the input data, allowing the model to focus on the most relevant features.

Key Components

1. Query (Q): Represents the information we want to focus on.
2. Key (K): Represents the data we will be looking at to find relevance.
3. Value (V): The actual information we will use after determining the relevance through attention weights.

Steps to Compute Attention Weights

1. **Compute the Dot Products**: Calculate the dot product between the query vector and each key vector to determine the compatibility score.

$$\text{score}_i = Q \cdot K_i$$

2. **Scale the Scores**: To stabilize gradients during training, scale the scores by dividing them by the square root of the dimension of the key vectors (denoted as d_k).

$$\text{scaled\_score}_i = \frac{\text{score}_i}{\sqrt{d_k}}$$

3. **Apply Softmax**: Apply the softmax function to the scaled scores to obtain the attention weights. This normalizes the scores into a probability distribution.

$$\text{attention\_weight}_i = \frac{\exp(\text{scaled\_score}_i)}{\sum_j \exp(\text{scaled\_score}_j)}$$

4. **Compute the Output**: Finally, compute the weighted sum of the value vectors using the attention weights.

$$\text{output} = \sum_i \text{attention\_weight}_i \cdot V_i$$

Continuation question : Okay great, now i want you to calculate the attention weights for the following data -

Let's say we have:

- Query $Q = [1, 0]$
- Key vectors $K_1 = [1, 0], K_2 = [0, 1]$
- Value vectors $V_1 = [2, 3], V_2 = [4, 5]$
- Dimension of the key vectors $d_k = 2$

Step 1: Compute the Dot Products

- $\text{score}_1 = Q \cdot K_1 = [1, 0] \cdot [1, 0] = 1$
- $\text{score}_2 = Q \cdot K_2 = [1, 0] \cdot [0, 1] = 0$

Step 2: Scale the Scores

$$\text{scaled\_score}_1 = \frac{1}{\sqrt{2}} \approx 0.707$$

$$\text{scaled\_score}_2 = \frac{0}{\sqrt{2}} = 0$$

Step 3: Apply Softmax

$$\text{attention\_weight}_1 = \frac{\exp(0.707)}{\exp(0.707) + \exp(0)} \approx \frac{2.025}{2.025 + 1} \approx 0.668$$

$$\text{attention\_weight}_2 = \frac{\exp(0)}{\exp(0.707) + \exp(0)} \approx \frac{1}{2.025 + 1} \approx 0.332$$

Step 4: Compute the Output

$$\text{output} = 0.668 \cdot V_1 + 0.332 \cdot V_2$$

$$= 0.668 \cdot [2, 3] + 0.332 \cdot [4, 5]$$

$$= [1.336, 2.004] + [1.328, 1.66] = [2.664, 3.664]$$

Q14. Explain the concept of multi-head attention in the Transformer architecture. How does it enhance the model's ability to capture different relationships within the input data?

Answer: Multi-head attention is a mechanism in the Transformer architecture that allows the model to focus on different parts of the input sequence simultaneously. Each attention head operates independently, enabling the model to capture various relationships and nuances within the data.

By processing information in parallel, multi-head attention enhances the model's ability to learn diverse representations. For example, one head might focus on syntactic relationships, while another emphasizes semantic meanings. This diversity allows for a richer understanding of context, which is crucial for tasks like machine translation.

Continuation question - Okay so in a Transformer model, we have the following hyperparameters:

- Embedding Size: 6
- Query Size (equal to Key and Value size): 3
- Number of Attention Heads: 2

1. **Understanding Dimensions**

   Given the above hyperparameters, calculate the shape of the Query (Q), Key (K), and Value (V) matrices after they have been passed through their respective Linear layers for a batch size of B .

2. **Attention Score Calculation**

   If the shape of the Q matrix after splitting into attention heads is (B, H, S, Q) where:

   - B is the batch size
   - H is the number of heads
   - S is the sequence length (let's assume $S = 4$)
   - Q is the query size per head

   Calculate the resulting shape of the Q matrix after performing the matrix multiplication $Q \cdot K^T$ to compute the attention scores, assuming K also has been split into heads.

3. **Merging Attention Scores**

   After computing the attention scores for each head, these scores need to be merged. Given that the shape of the attention scores for each head is (B, H, S, Q), what will be the shape of the merged attention score matrix after collapsing the head dimension?

1. **Understanding Dimensions**

   The shape of the Q, K, and V matrices after the Linear layers is $(B, S, 6)$. After splitting for 2 heads, each will have:

   - Q: $(B, S, 3)$
   - K: $(B, S, 3)$
   - V: $(B, S, 3)$

2. **Attention Score Calculation**

   The shape of the Q matrix after splitting is $(B, 2, 4, 3)$ because:

   - $H = 2$
   - $S = 4$
   - $Q = 3$

   When computing $Q \cdot K^T$, where K is also $(B, 2, 4, 3)$, the resulting shape of the attention scores would be:

   $$\text{Attention Scores shape} = (B, H, S, S) \quad (4 \text{ for both Q and K})$$

3. **Merging Attention Scores**

   The merged attention score matrix shape after collapsing the head dimension will be:

   $$(B, S, H \times Q) = (B, S, 2 \times 3) = (B, S, 6)$$

Q15. In a comparison between a 20-layer CNN and a 56-layer CNN, the authors found that the deeper model performed worse due to vanishing/exploding gradients. Explain how ResNet's skip connections improve the performance of deeper networks and what the key difference is compared to the traditional CNN architectures.

Answer: When deep neural networks have too many layers, they face the vanishing/exploding gradient problem. This causes gradients to either become too small (vanishing) or too large (exploding), making it difficult for the model to converge during training. ResNet solves this issue by introducing residual blocks, which use skip connections to bypass certain layers. This helps the model by allowing layers to learn residual functions rather than learning the full transformation, making it easier to train very deep networks.

With ResNet, the skip connections allow deeper models (like 56-layer networks) to perform better because they enable gradients to pass through layers without being diminished. This is a key difference from traditional architectures, which require each layer to directly learn transformations, making it harder to train deeper networks. ResNet effectively mitigates this problem by focusing on learning residuals.

Q16. - Explain the main concept behind Highway Networks. What role do the Transform Gate (T) and Carry Gate (C) play in controlling the flow of information through the network?

Answer: Highway Networks are a type of neural network that introduce gated pathways to control the flow of information through layers. These networks use two gates:

1. Transform Gate (T) – controls how much of the transformed input is passed to the next layer.
2. Carry Gate (C) – allows the raw input to bypass the transformation and pass directly to the next layer.

The key idea is that instead of transforming all inputs through non-linear functions like in traditional networks, Highway Networks decide whether to transform or directly pass the input, making it easier to train very deep networks by overcoming problems like vanishing gradients.

Q17. Explain the primary purpose of using 1x1 convolutions in the Inception Network (GoogLeNet) architecture. How does this technique contribute to both performance improvement and computational efficiency?

Answer: The 1x1 convolution in the Inception Network (GoogLeNet) is primarily used for dimensionality reduction. It reduces the number of input channels, which drastically cuts down the computational cost before applying more expensive convolutions like 3x3 or 5x5. This helps improve performance by reducing the number of parameters and computation required without losing significant information, allowing for deeper networks while avoiding overfitting and minimizing resource consumption.

Q18. Consider an image of size 14×14×480 where you need to apply a 5x5 convolution. Compare the computational cost (in terms of operations) when using 1x1 convolutions before the 5x5 convolution to reduce dimensionality from 480 to 16 channels, versus performing the 5x5 convolution directly without dimensionality reduction. How much computational saving is achieved by using 1x1 convolutions?

- **Without using 1x1 convolution:** The number of operations for performing a **5x5 convolution** directly on an image of size $14 \times 14 \times 480$ is:

$$\text{Operations} = 14 \times 14 \times 480 \times (5 \times 5) = 112.9 \text{ million operations.}$$

- **Using 1x1 convolution:** First, apply **1x1 convolution** to reduce the input from 480 channels to 16 channels:

$$\text{1x1 Operations} = 14 \times 14 \times 16 \times (1 \times 1 \times 480) = 1.5 \text{ million operations.}$$

Then, apply the **5x5 convolution** on the reduced 16 channels:

$$\text{5x5 Operations} = 14 \times 14 \times 48 \times (5 \times 5 \times 16) = 3.8 \text{ million operations.}$$

Total operations with 1x1 convolutions:

$$1.5 \text{ million} + 3.8 \text{ million} = 5.3 \text{ million operations.}$$

- **Computational savings:** By using the 1x1 convolution, the reduction in operations is:

$$112.9 \text{ million} - 5.3 \text{ million} = 107.6 \text{ million operations.}$$

This demonstrates a significant reduction in computational cost, making the process much more efficient.

Q19. What is the core concept behind EfficientNet, and how does the compound scaling method improve its performance and computational efficiency compared to previous CNN architectures?

Answer: EfficientNet is a family of CNNs that uses a novel compound scaling method to uniformly scale the network's width, depth, and resolution. Unlike previous architectures,

which scaled these dimensions independently, EfficientNet scales them together using a fixed compound coefficient. This approach improves both the model's performance and computational efficiency, allowing EfficientNet to achieve high accuracy while using fewer resources. The compound scaling ensures that the network grows in a balanced manner, preventing overfitting and excessive computational costs.

Q20. Consider two hidden layers of size 224×224 with C1 and C2 channels, respectively, connected by a 3×3 convolutional layer. Describe how to initialize the weights using He initialization.

Answer: Understanding He Initialization

He initialization is particularly effective for layers using ReLU activation functions. The purpose of this technique is to prevent the vanishing or exploding gradients problem by setting the weights to small random values that are appropriately scaled.

Calculating the Number of Input Units

For a convolutional layer, the number of input units nnn is determined by the number of input channels and the convolutional kernel size. In this case, since we have a 3×3 convolutional kernel, we calculate nnn as:

n=C1×3×3

where C1 is the number of channels from the previous layer.

Applying He Initialization

To initialize the weights, we sample from a normal distribution centered at zero, with a standard deviation defined by:

σ= root(2/n) = root(2/C1×9)

So, the weights W would be initialized as:

W~N(0,σ^2)

Q21. Consider a 2D convolutional layer with kernel size 5×5 that takes 3 input channels and returns 10 output channels. How many convolutional weights are there? How many biases?

Answer: Number of weights=(Kernel height×Kernel width×Number of input channels)×Number of output channels

Given:

- Kernel size = 5×5
- Number of input channels = 3
- Number of output channels = 10

We can substitute these values into the formula:

Number of weights=750

Number of Biases:

The number of biases in a convolutional layer is equal to the number of output channels because each output channel has one bias term. Therefore:

Number of biases = Number of output channels = 10

Q22. Consider a 1D convolutional network where the input has three channels. The first hidden layer is computed using a kernel size of three and has four channels. The second hidden layer is computed using a kernel size of five and has ten channels. How many biases and how many weights are needed for each of these two convolutional layers?

Answer: First Hidden Layer

- Input channels: 3
- Kernel size: 3
- Output channels: 4

Number of Weights:

Number of weights=Kernel size×Input channels×Output channels = 3×3×4=36

Number of Biases: The number of biases is equal to the number of output channels = 4

Second Hidden Layer

- Input channels: 4 (from the first layer)
- Kernel size: 5
- Output channels: 10

Number of Weights:

Number of weights=Kernel size×Input channels×Output channels = 5×4×10=200

Number of Biases:

Number of biases=10

Q23. A network consists of three 1D convolutional layers. At each layer, a zero-padded convolution with kernel size three, stride one, and dilation one is applied. What size is the receptive field of the hidden units in the third layer?

Answer: In a 1D convolutional network with a zero-padded convolution using kernel size 3, stride 1, and dilation 1, the receptive field size can be calculated as follows:

1. Layer 1:
    - Kernel size: 3
    - Stride: 1
    - Receptive field size: 3
2. Layer 2:
    - Kernel size: 3
    - Stride: 1
    - Receptive field size: 3+(3−1)=5

3. Layer 3:

    - Kernel size: 3
    - Stride: 1

- Receptive field size: 5+(3−1)=7

Thus, the receptive field size of the hidden units in the third layer is 7.

Q24. A network consists of three 1D convolutional layers. At each layer, a zero padded convolution with kernel size seven, stride one, and dilation one is applied. What size is the receptive field of hidden units in the third layer?

Answer: In this scenario, we will compute the receptive field sizes for three 1D convolutional layers, each with kernel size 7, stride 1, and dilation 1:

1. Layer 1:
    - Kernel size: 7
    - Stride: 1
    - Receptive field size: 7

2. Layer 2:

    - Kernel size: 7
    - Stride: 1
    - Receptive field size: 7+(7−1)=13

3. Layer 3:

    - Kernel size: 7
    - Stride: 1
    - Receptive field size: 13+(7−1)=19

Thus, the receptive field size of the hidden units in the third layer is 19.

Q25. Consider a convolutional residual block that contains a batch normalization operation, followed by a ReLU activation function, and then a 3×3 convolutional layer. If the input and output both have 512 channels, how many parameters are needed to define this block? Now

consider a bottleneck residual block that contains three batch normalization/ReLU/convolution sequences. The first uses a 1×1 convolution to reduce the number of channels from 512 to 128. The second uses a 3×3 convolution with the same number of input and output channels. The third uses a 1×1 convolution to increase the number of channels from 128 to 512. How many parameters are needed to define this block?

Answer: Convolutional Residual Block

First Convolutional Layer (3x3 convolution):

- Input channels = 512
- Output channels = 512
- Kernel size = 3x3
- Number of parameters = (kernel height × kernel width × input channels + 1 (bias)) × output channels
- Number of parameters = (3×3×512+1)×512=(9×512+1)×512=(4608+1)×512=4609×512=2,359,488

Batch Normalization:

- Each batch normalization layer has 2 parameters per channel (gamma and beta).
- For 512 channels, the number of parameters = 2×512=1024

ReLU Activation:

- ReLU does not have any parameters.

Total Parameters for the Convolutional Residual Block:

Total=Convolutional Layer+Batch Norm=2,359,488+1024=2,360,512

Bottleneck Residual Block

This block consists of three sets of operations.

1. First Sequence (1x1 convolution):
    - Input channels = 512

- Output channels = 128
- Number of parameters = (1×1×512+1)×128=(512+1)×128=513×128=65,664

2. Second Sequence (3x3 convolution):
    - Input channels = 128
    - Output channels = 128
    - Number of parameters = (3×3×128+1)×128=(9×128+1)×128=(1152+1)×128=1153×128=147,584

3. Third Sequence (1x1 convolution):
    - Input channels = 128
    - Output channels = 512
    - Number of parameters = (1×1×128+1)×512=(128+1)×512=129×512=66,048

4. Batch Normalization:

    - Each batch normalization layer has 2 parameters per channel.
    - For 512 channels (third sequence) and 128 channels (first and second sequences):
        - First sequence: 2×128=256
        - Second sequence: 2×128=256
        - Third sequence: 2×512=1024

Total Batch Normalization Parameters:

256+256+1024=1536

Total Parameters for the Bottleneck Residual Block:

Total=First Sequence+Second Sequence+Third Sequence+Batch Norm=65,664+147,584+66,048+1536=280,832

Summary

- Convolutional Residual Block: 2,360,512 parameters.
- Bottleneck Residual Block: 280,832 parameters.

NLP

Q1. What is Tokenization?

Answer: Tokenization is the process of breaking down text into individual units called tokens. These tokens could be words, sentences, or subwords depending on the application. For instance, breaking a sentence into words is word-level tokenization

Q2. What is Lemmatization?

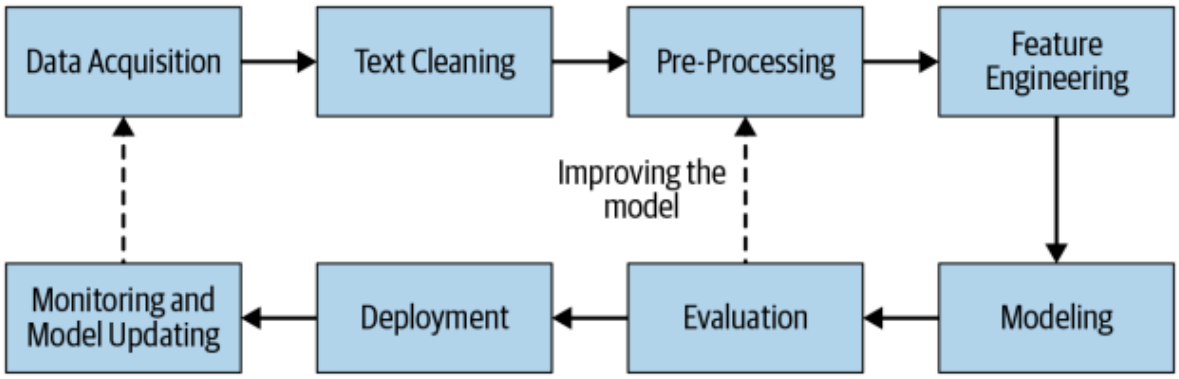
Answer: Lemmatization reduces words to their base or root form, known as a lemma. Unlike stemming, lemmatization considers the meaning of the word. For example, the lemmatizer reduces "running" to "run" based on its part of speech.

Q3. What is Word2Vec? How does it work?

Answer: Word2Vec is a neural network-based approach that represents words in continuous vector space where semantically similar words are mapped to nearby points. It works by using either Continuous Bag of Words (CBOW) or Skip-gram models, predicting context words given a word (CBOW) or predicting a word given its context (Skip-gram).

Q4. Show the generic pipeline to build a modern day, data driven NLP system.

Answer-



Q5. How is Lemmatization different from Stemming? Give an example.

Answer- Lemmatization is mapping of various forms of a word to its base word whereas in stemming we remove the suffixes from a word so that it is reduced to its base form. Lemmatization needs greater linguistic knowledge. For example, “better” after stemming is “better” but “better” after lemmatization is “good”.

Stemming	Lemmatization
adjustable -> adjust	was -> (to) be
formality -> formaliti	better -> good
formaliti -> formal	meeting -> meeting
airliner -> airlin	

Q6. For Tokenization, we mostly use NLTK and SpaCy. When will you use what and why?

Answer- NLTK is more complex and requires more code to achieve some tasks but it is highly customizable and one can experiment with different algorithms. Whereas, SpaCy is designed for practical application and also it is more user friendly and quick to implement.
Prefer using SpaCy when we need a fast and reliable NLP library for real world applications and when we want to implement NLP tasks quickly and efficiently. NLTK can be preferred in research i.e. when you need to experiment with different models and algorithms.

Q7. What is Data Augmentation? How is it done in NLP projects?

Answer- Data Augmentation is taking a small dataset and use that in order to create more data. Some ways are- Replacing entities, TF-IDF–based word replacement, Adding noise to data, Back translation, Synonym replacement, Bigram flipping

Q9. What is Named Entity Recognition(NER)?

Answer- Named Entity Recognition helps in detection and categorization of proper names in a text into specific categories such as people, organisations, geographical markers, time elements, etc.

NER helps in extracting structured information from unstructured text, making it easier to analyse and understand. It is used in applications like information retrieval, question answering, and more.

Q10. What is POS Tagging in NLP? What are some challenges related to POS Tagging?

Answer- Part-of-Speech (POS) Tagging plays a fundamental role by identifying the grammatical components of text, such as words and phrases, and labelling them with their corresponding parts of speech.
Some of the challenges are:

1. Depending on the use or word or their context, parts of speech can differ.
2. Some words can have more than 1 POS. For example, the word “well” can be a noun, adverb or an adjective.

Q11. Explain cosine similarity and its use in NLP?

Answer- Cosine similarity measures the similarity between two vectors by calculating the cosine of the angle between them. In natural language processing (NLP), it's often used to assess the similarity between text documents or word embeddings.It's used to compare text documents in NLP by analysing word usage. It also helps to find similar words in models like Word2Vec through vector comparison.

Q12. Explain the Bag of Words model and what are its limitations?

Answer- The Bag-of-Words (BoW) model is a simple and widely used method in NLP for representing text data. It involves converting text into a vector of word frequencies, ignoring grammar and word order.
Each unique word in the text corpus becomes a feature, and the vector represents the count of each word in a document. This model is used for tasks like text classification and clustering.

Bag of Words (BoW) has some limitations that can affect its usefulness in certain tasks. It ignores the order of words, which means it loses the flow of how words are used together—something that’s important for tasks like language modelling or sentiment analysis. It also tends to create large, sparse matrices when dealing with big vocabularies, which can

demand a lot of memory and processing power. Plus, BoW treats all words as separate from each other, so it misses the context they appear in, and it gives equal importance to every word, even if some aren't very meaningful. Lastly, BoW struggles with words it hasn't seen before, making it less effective when dealing with new or rare words.

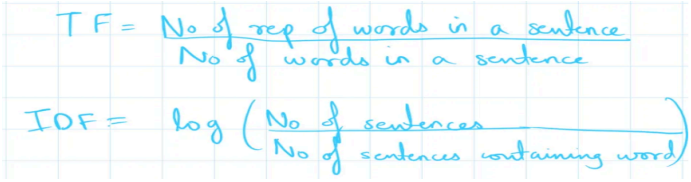
Q13. What are Text N- Grams in NLP?

Answer- N-grams are sequential word or character sets, with "n" indicating the number of elements in a particular set. They play a crucial role in understanding context and text prediction, especially in statistical language models.

Q14. Explain TF-IDF in NLP.

Answer- It stands for Term Frequency - Inverse Document Frequency. The idea behind it is to quantify the importance of a term in a document with respect to its frequency in the document and its rarity across multiple documents.

TF-IDF= TF * IDF



Q15. Why should we remove Stop Words from our corpus? Are there any cases when we should not really remove them?

Answer-Stop words are words that are commonly used but have little to no value in helping processors answer queries, for example - a,an,the,not,but,or,and etc. They provide no meaningful information, especially if we are building a text classification model. Therefore, we have to remove stopwords from our dataset.

As the frequency of stop words are too high, removing them from the corpus results in much smaller data in terms of size.

But there are some cases where we should not remove the stop words.

Like- Language Translations, Chatbots, basically any case where any valuable data might be lost.

Q16. What is the advantage of using Word Embedding Techniques over other models?

Answer-

1. Similar words have similar vectors.

2. The Dimensions of those vectors are low.

Q17. What are the steps of Sentimental Analysis of social media posts?

Answer-

1. Data collection: Gather a diverse dataset of social media posts with sentiment labels.
2. Preprocessing: Handle social media-specific elements like hashtags, @mentions, emojis, and slang.
3. Feature extraction: Use techniques like word embeddings or TF-IDF, potentially incorporating social media-specific features.
4. Model selection: Choose an appropriate model (e.g., Naive Bayes, SVM, or deep learning models like LSTM or BERT).
5. Training and evaluation: Train the model and evaluate using metrics like accuracy, F1-score, and confusion matrix.
6. Handling challenges: Address issues like sarcasm detection, mixed sentiments, and context-dependent sentiments.

Q18. The above models are trained on what type of data and what are its training objectives?

Model	Training Data	Training Objectives
BERT	BooksCorpus, English Wikipedia	- Masked Language Modeling (MLM) - Next Sentence Prediction (NSP)
RoBERTa	BooksCorpus, English Wikipedia, Common Crawl	- Masked Language Modeling (MLM) (dynamic masking)
GPT	Diverse internet text (OpenAI WebText)	- Unidirectional Language Modeling (predict next word)
T5	Various datasets across NLP tasks	- Text-to-Text format for all tasks (masked language modelling)
BART	CNN/Daily Mail, SQuAD, ConvAIR 2	- Denoising Autoencoder (reconstruct original text from corrupted input)

LLama	Publicly available datasets from diverse domains	- Causal Language Modelling (predict next token)

Q19. When should one use which model and what are the respective limitations of these models?

Answer-

BERT:

When to Use: BERT is fantastic for tasks like text classification, sentiment analysis, named entity recognition, and question answering. If one needs to understand the context of text or categorise it, BERT is a solid choice.

When Not to Use: BERT might not be a good fit for something that requires deep common-sense reasoning or creative writing.

Limitations:BERT struggles with common-sense reasoning and inference. It also doesn't adapt well to new tasks without fine-tuning and it can pick up biases from the data it was trained on.

RoBERTa:

When to Use: RoBERTa is an upgraded version of BERT. It works well for the same tasks but often performs better because of its improved training methods.

When Not to Use: Just like BERT, it's not great for tasks that need deep reasoning or creative generation.

Limitations: RoBERTa has the same limitations as BERT regarding common-sense reasoning and it is also resource-intensive when it comes to training.

GPT:

When to Use: GPT is good for text generation tasks.

When Not to Use: It is not suitable for tasks that require structured output like classifications or tasks that require factual accuracy.

Limitations: It sometimes generates text that is easy to believe in but isn’t true.

T5:

When to Use: T5 treats every task as a text-to-text problem. It’s excellent for translation and summarization.

When Not to Use: Not suitable for tasks that don't fit into the text-to-text framework.

Limitations: It requires a lot of computational resources for training. Also, its performance can dip on very specialised tasks unless it is fine-tuned properly.

BART:

When to Use: BART is great for generating text and summarising information. It’s particularly useful when input data is corrupted or jumbled.

When Not to Use: Not suitable if one needs strict factual accuracy in the outputs.

Limitations: The complexity of BART's architecture can lead to longer training times. Additionally, it may struggle with maintaining logicality in long-form content.

LLaMA:

When to Use: LLaMA serves as a general-purpose model for understanding and generating language.

When Not to Use:It may not perform well on highly specialized tasks without enough fine-tuning.

Limitations:LLaMA can be less transparent in how it makes decisions and also inherent biases from its training data.

Q20. What will be the output of an encoder with 1024 input points?

Answer- There will be 1024 output points but the information will change in output with respect to the input. The information will change based on how an encoder processes the input. The encoder transforms the input data into new representations that capture important relationships and patterns. Each output point reflects not just the original input but

also its context, making it easier to understand complex information. So, while the number of points stays the same, the information they carry is richer and more meaningful.

Q21. How will you predict the next word using RNN and what are its limitations?

Answer-

Limitations:
1.Vanishing gradient problem: One of the significant drawbacks of basic RNNS is the vanishing gradient problem. It occurs when gradients during training become extremely small as they are backpropagated through time This limits the network's ability to capture long-range dependencies

2. Exploding gradient problem: RNNs can also suffer from the exploding gradient problem, where gradients become exceptionally large during training, causing numerical instability. Exploding gradient easier to detect and manage

3. Limited Memory: Traditional RNNs have a limited memory capacity, and they struggle to carry information across many time steps. This can be problematic when dealing with long sequences where network may "forget" important information from earlier time steps

4. Biased Toward Recent Data. Lack of Global Context: Following from the above point, in an RNN, as data progresses over time steps, the influence of past data diminishes. This means the network can become biased toward more recent data in the sequence and struggle to capture global context.

5. Difficulty with Parallelization: RNNs process data sequentially, which makes parallelization challenging, leading to slower training. As a result, RNNs are not able to take complete advantages of modern hardware architectures such as GPUs designed for parallel processing.

Q22. Explain why positional embeddings are added to word embeddings?

Answer- Positional embeddings are added to the word embeddings to provide information about the relative or absolute position of the tokens in the sequence. Without positional embeddings, the model would treat permutations of the same set of words as identical. For example, “The quick brown fox” would be indistinguishable from “fox brown quick The”. Positional embeddings resolve this by providing a unique representation for each position in

the sequence, allowing the model to differentiate “The” as the first word from “fox” as the last.

Q23. What is Attention Mechanism? Why is it important?

Answer- Attention mechanism helps models to focus on the most relevant portion of the input data. The attention model in NLP searches for the most relevant information in the source sentences. Before attention mechanisms came along, models like RNNs and LSTMs tried to understand sentences by converting all the information into one small summary. This often meant they missed important details, especially in longer sentences. In the world of images, techniques like saliency maps aimed to point out what was important, but they weren’t very flexible. Essentially, these earlier methods struggled to keep track of what really mattered. The breakthrough with attention mechanisms was inspired by how we humans focus on specific things while tuning out the noise around us, making it much easier for computers to understand and recognize what’s important in both language and visuals.

1. Attention mechanism improves model performance by enabling them to focus on relevant parts of the input sequence.
2. Reduces workload by breaking down lengthy sequences into smaller manageable components.
3. Improves decision-making by enhancing the interpretability of the AI and NLP models.

Q24. What happens if there are more attention heads?

Answer-

1. As each attention head can pay attention to different parts of the input it makes the model understand more details and relationships in the text.
2. If there’s only one attention head, some words might get too much focus. More heads help spread attention evenly across all words, making sure everything is considered.
3. More heads mean more calculations, which can slow things down and require more computer power. So, one needs to find the right balance.

Q25. Can one use encoders from one model and decoders from another model?

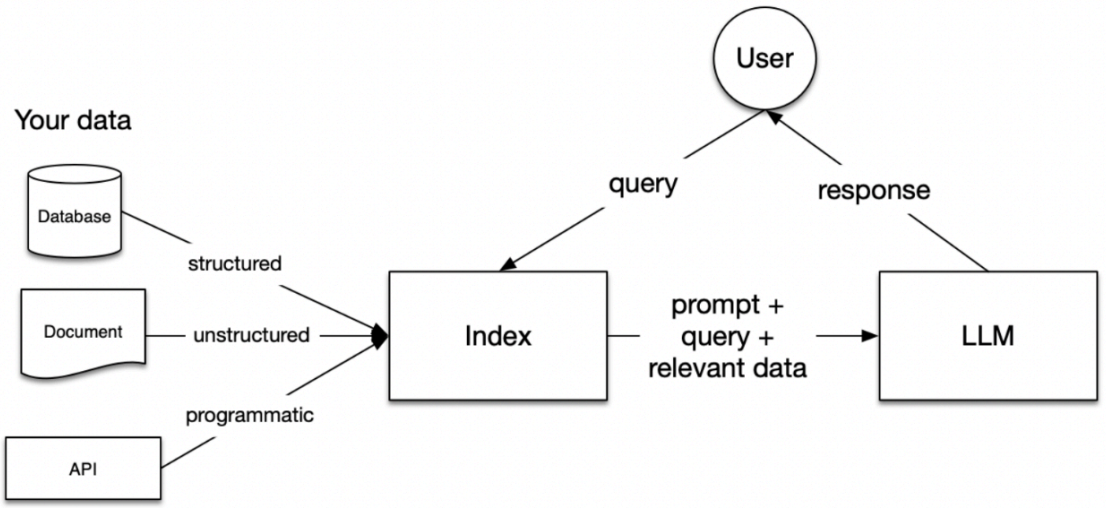
Answer- No. The size might be different. Even if the size is the same, they must have been trained on different datasets.

Q26. You will prefer using what among Transformers, RNNs or traditional models if you have limited data?

Answer- If one has limited data, it's generally better to use RNNs (Recurrent Neural Networks) or traditional models rather than transformers. RNNs can effectively handle sequential data and are more suitable for smaller datasets because they require fewer parameters and are

less prone to overfitting compared to transformers, which need large amounts of data to perform well due to their complexity. Traditional models, like logistic regression or decision trees, are also good options for small datasets as they are simpler and easier to interpret. Therefore, starting with RNNs or traditional models allows you to achieve reasonable performance without the risk of overfitting that comes with using more complex transformer architectures.

Q27. Explain the complete pipeline of RAG?



Q28. What are the main challenges you face working in NLP?

Answer: One of the main challenges in NLP is accurately capturing semantics—understanding the meaning behind words, phrases, and sentences, especially with things like idioms and metaphors. Ambiguity is another big issue since many words or phrases can have multiple meanings depending on the context. Context itself is critical for accurate interpretation, especially when resolving references like pronouns. Additionally, the diversity of languages and dialects presents a challenge, particularly with low-resource languages. Data limitations and biases can impact both performance and fairness, and lastly, integrating real-world knowledge and common sense into NLP models is still an ongoing difficulty.

Q29. What do you mean by Text Augmentation in NLP and what are the Text Augmentation techniques used in NLP?

Answer: Text Augmentation in NLP refers to the process that generates new or modified textual data from existing data in order to increase the diversity and quantity of training samples. Text augmentation techniques apply numerous alterations to the original text while keeping the underlying meaning.

Different text augmentation techniques in NLP include:

1.Synonym Replacement: Replacing words in the text with their synonyms to introduce variation while maintaining semantic similarity.

2.Random Insertion/Deletion: Randomly inserting or deleting words in the text to simulate noisy or incomplete data and enhance model robustness.

3. Word Swapping: Exchanging the positions of words within a sentence to generate alternative sentence structures.

4.Back translation: Translating the text into another language and then translating it back to the original language to introduce diverse phrasing and sentence constructions.

5. Random Masking: Masking or replacing random words in the text with a special token, akin to the approach used in masked language models like BERT.

6. Character-level Augmentation: Modifying individual characters in the text, such as adding noise, misspellings, or character substitutions, to simulate real-world variations.

7. Text Paraphrasing: Rewriting sentences or phrases using different words and sentence structures while preserving the original meaning.

8. Rule-based Generation: Applying linguistic rules to generate new data instances, such as using grammatical templates or syntactic transformations.

Q30. Why do RNNS face a vanishing gradient problem?

Answer: Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks face the vanishing gradient problem, which significantly impacts their ability to learn from long sequences of data. This issue arises primarily due to the way gradients are propagated through these networks during backpropagation.

Vanishing Gradient Problem

Propagation of Gradients: In RNNs, as gradients are passed backward through time steps, they are multiplied by weights at each step. If these weights are less than one, repeated multiplication can lead to gradients that become exceedingly small, effectively vanishing. This diminishes the model's ability to update weights meaningfully, especially for earlier time steps in the sequence.

Activation Functions: The choice of activation functions also contributes to this problem. Functions like sigmoid or tanh squash input values into a limited range (0 to 1 for sigmoid), which can further compress gradients, leading to very small values during backpropagation.

Long Sequences: The vanishing gradient problem becomes more pronounced with longer sequences. As the network attempts to learn from distant time steps, the cumulative effect of multiplying small gradients can render earlier gradients negligible, causing the network to forget crucial information from earlier inputs

Q31. How do Transformers ensure that there aren't any vanishing gradient related problems?

Answer: Transformers effectively mitigate the vanishing gradient problem through several architectural features that enhance gradient flow during training. Here are the key mechanisms that contribute to this robustness:

Key Mechanisms
Self-Attention Mechanism:
In Transformers, each token in a sequence can directly attend to every other token, regardless of their position. This contrasts with RNNs, where information must pass through each time step sequentially. As a result, gradients can flow more directly between tokens, reducing the risk of vanishing gradients when learning long-range dependencies.

Residual Connections:
Transformers incorporate residual (or skip) connections that allow gradients to bypass certain layers during backpropagation. This design helps maintain gradient magnitude by providing alternative pathways for gradient flow, thus preventing them from diminishing too much as they propagate through multiple layers.

Layer Normalization:
Layer normalization is employed in Transformers to stabilize activations across layers, which helps prevent gradients from becoming excessively small or large. By normalizing the inputs to each layer, this technique ensures that the training process remains stable and efficient.

Pre-Layer Normalization:
Recent studies indicate that using pre-layer normalization can further alleviate issues related to gradient vanishing. This approach normalizes inputs before they enter the attention and feed-forward layers, which has been shown to improve convergence and stability during training

Multi-Head Attention:
The multi-head attention mechanism allows the model to focus on different parts of the input sequence simultaneously, enhancing its ability to capture complex patterns without suffering from diminishing gradients. Each head can learn different relationships in the data, contributing to a richer representation

Q32. In Transformers why do we need to create 3 different vectors(query key value) for each embedding before attention is calculated?

Answer: Purpose of Q, K, and V Vectors
Different Roles in Attention Mechanism:
Query: Represents the information that a particular token seeks from other tokens in the sequence. It essentially asks, "What am I looking for?"
Key: Acts as a descriptor for each token, indicating what features or attributes it possesses. This can be thought of as answering, "What can I offer?"
Value: Contains the actual information or content that will be returned based on the relevance determined by the Query and Key. It answers, "What is my value in this context?"

Facilitating Contextualization:
By separating these roles, the model can compute attention scores based on how well each Query aligns with the Keys of other tokens. This process allows for a nuanced understanding of which tokens should contribute to the final representation of a given token based on its context within the sequence

Linear Transformations:
Each of these vectors is derived from the same input embedding through different linear transformations using distinct weight matrices (Wq for Queries, Wk for Keys, and Wv for Values). This allows the model to learn different projections of the input data that capture various aspects relevant to attention computation

Parallel Computation:
The design allows all tokens in a sequence to be processed simultaneously rather than sequentially. This parallelism is crucial for efficiency and scalability in training large models on extensive datasets

Q33. Explain how BERT performs Masked Language Modelling?

Answer:
BERT (Bidirectional Encoder Representations from Transformers) employs a technique known as Masked Language Modeling (MLM) during its training process. This method is crucial for enabling BERT to learn contextual representations of words based on their surrounding context. Here’s how MLM works in detail:
Mechanism of Masked Language Modeling
Token Masking:

During training, 15% of the tokens in each input sequence are randomly selected to be masked. The purpose of this masking is to create a scenario where the model must predict the original token based on the context provided by the other, non-masked tokens in the sequence.

Replacement Strategy:
The selected tokens undergo a specific replacement strategy:80%
This strategy helps ensure that the model does not become overly reliant on the [MASK] token and maintains a robust understanding of context from both masked and unmasked words.

Contextual Prediction:
After masking, BERT processes the entire sequence through its transformer architecture, which uses self-attention mechanisms to consider both left and right contexts simultaneously. This bidirectional approach allows BERT to capture nuanced relationships between words effectively.
The model then attempts to predict the masked tokens based on their context within the sequence. Each output vector from BERT's encoder is transformed into a probability distribution over the vocabulary using a classification layer.

Loss Calculation:
The model's predictions for the masked tokens are evaluated against their actual values to compute a loss. The objective during training is to minimize this loss, thereby improving BERT's ability to accurately predict masked words based on context.

Q34. Why don’t Transformers use Batch Normalization and explain the type of normalization technique that they use?

Answer: Transformers do not use Batch Normalization primarily due to the nature of their architecture and the sequence-based tasks they are designed for. Instead, they utilize Layer Normalization, which is more suitable for their operational framework. Here’s a detailed explanation of why Batch Normalization is less effective in Transformers and how Layer Normalization is implemented:
Reasons for Not Using Batch Normalization
Sequential Data Handling:
Transformers process sequences of data (like sentences) in parallel, rather than in batches where the entire sequence is processed at once. Batch Normalization computes statistics (mean and variance) across a batch of samples, which can introduce noise and instability when applied to individual sequences or variable-length inputs.
Dependence on Sequence Length:

In NLP tasks, sequences can vary significantly in length. Batch Normalization relies on consistent batch sizes and can lead to issues when the model encounters sequences of different lengths, as the normalization statistics become less reliable.
Training Dynamics:
The use of Batch Normalization can complicate the training dynamics in Transformers, especially with respect to learning rates and convergence. It requires careful tuning of hyperparameters and can lead to instability during training, particularly in deep architectures.

Q35. What's the difference between Self Attention, Multi-Head Attention and Cross-Attention?

Self-attention, multi-head attention, and cross-attention are key components of the Transformer architecture, each serving distinct purposes in processing input sequences. Self-attention allows each token in an input sequence to attend to all other tokens within the same sequence, enabling the model to capture relationships and dependencies regardless of their positions. This mechanism enhances the contextual representation of each token based on its surrounding context, which is crucial for tasks like language modeling and understanding sentence structure. For example, in a sentence like "The cat sat on the mat," self-attention helps the model understand how "cat" relates to "sat" and "mat," thereby capturing the overall meaning.

Multi-head attention extends self-attention by allowing multiple attention heads to operate in parallel. Each head computes its own attention scores using separate sets of learned parameters, which allows the model to jointly attend to information from different representation subspaces at various positions. This capability is particularly beneficial in tasks like translation, as it enables the model to focus on different parts of the source sentence simultaneously when generating each word in the target sentence.

Cross-attention, on the other hand, allows one sequence (typically the target) to attend to another sequence (usually the source). In this mechanism, the Query comes from one sequence while the Key and Value come from another. Cross-attention is essential for tasks where outputs depend on multiple inputs, such as machine translation or image captioning. For instance, while generating a translation for a word in the target language, cross-attention enables the model to look back at relevant words in the source language sentence. In summary, self-attention focuses on relationships within a single sequence, multi-head attention enhances this by using multiple perspectives simultaneously, and cross-attention connects two different sequences for more complex tasks.

Q36. What is the difference between GPT and BERT in terms of generative tasks?

Answer: The primary difference between GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers) in terms of generative tasks lies in their architectures and training methodologies, which influence how they generate language. GPT is designed as a decoder-only model that generates text in a unidirectional manner, meaning it predicts the next word based solely on the preceding context. This capability makes GPT particularly adept at generative tasks, such as writing coherent paragraphs, completing sentences, or creating dialogue, as it generates text sequentially, word by word.

In contrast, BERT employs a bidirectional architecture that processes text by considering both the left and right contexts simultaneously. While BERT is primarily focused on understanding and analyzing language rather than generating it, it uses a technique called Masked Language Modeling during training, where certain words in a sentence are masked, and the model learns to predict these masked words based on their surrounding context. This approach enables BERT to excel in tasks that require deep contextual understanding, such as sentiment analysis and question answering. However, it is not inherently designed for generating text in the same way that GPT is. Overall, while both models leverage the transformer architecture and demonstrate high performance in various NLP tasks, GPT's generative capabilities are more pronounced due to its unidirectional approach and focus on sequential text generation. In contrast, BERT's strength lies in its ability to comprehend and analyze language contextually, making it more suitable for tasks that require understanding rather than generation.

Q37. In the Transformer architecture which part of the model(encoder or decoder) is autoregressive and which part of it is non-autoregressive at run time and how?

Answer: In the Transformer architecture, the encoder and decoder serve different roles regarding autoregressive and non-autoregressive processes during runtime. The decoder is autoregressive, meaning it generates output sequences one token at a time in a sequential manner. This is achieved through a mechanism called causal masking, which prevents the decoder from accessing future tokens when predicting the next token in the sequence. As a result, each generated token relies on the previously generated tokens, allowing for coherent and contextually relevant text generation.

On the other hand, the encoder operates in a non-autoregressive manner. It processes the entire input sequence simultaneously, allowing it to capture relationships and dependencies among all tokens without the constraints of sequential generation. The encoder's ability to attend to all input tokens at once facilitates a comprehensive understanding of the input context, which is then utilized by the decoder during its autoregressive generation process.

This distinction between autoregressive and non-autoregressive functionalities within the Transformer architecture enables efficient handling of various tasks. While the encoder focuses on understanding and encoding input data, the decoder generates output in a controlled and sequential manner, ensuring that each step builds upon the previous context.

Q38. Write Python code using gensim to train Word2Vec embeddings on a custom text corpus.

```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from nltk.corpus import stopwords
import nltk

# Sample text corpus
corpus = [
    "Word embeddings are a type of word representation that allows words to be represented as vectors in a continuous vector space.",
    "Word2Vec is a popular algorithm used to produce word embeddings.",
    "Gensim is a Python library that can be used for natural language processing and building word embeddings.",
    "Training word embeddings requires a large amount of text data to be effective."
]

# Preprocess the corpus: tokenize and remove stopwords
def preprocess(text):
    stop_words = set(stopwords.words("english"))
    return [word for word in simple_preprocess(text) if word not in stop_words]

# Tokenize and clean each sentence in the corpus
processed_corpus = [preprocess(sentence) for sentence in corpus]

# Initialize and train the Word2Vec model
```

```
model = Word2Vec(sentences=processed_corpus, vector_size=100, window=5,
min_count=1, workers=4, sg=1)

# Save the model
model.save("word2vec_model.model")

# Example usage: find most similar words
similar_words = model.wv.most_similar("word", topn=5)
print("Words similar to 'word':", similar_words)
```

Q39. How would you use pre-trained word embeddings like Word2Vec or GloVe in a neural network model using PyTorch or TensorFlow?

Answer:

1. PyTorch Implementation

```
import torch
import torch.nn as nn
from gensim.models import KeyedVectors

# Load pre-trained Word2Vec or GloVe embeddings
embedding_path = 'path/to/your/word2vec_or_glove_file'  # e.g.,
'GoogleNews-vectors-negative300.bin'
word_vectors = KeyedVectors.load_word2vec_format(embedding_path,
binary=True)

# Prepare the embedding matrix
embedding_dim = word_vectors.vector_size
vocab_size = len(word_vectors.index_to_key)
embedding_matrix = torch.zeros((vocab_size, embedding_dim))

# Map words to index in embedding matrix
word_to_idx = {word: idx for idx, word in
enumerate(word_vectors.index_to_key)}

# Fill embedding matrix with pre-trained embeddings
for word, idx in word_to_idx.items():
    embedding_matrix[idx] = torch.tensor(word_vectors[word])

# Define a simple model with embedding layer using PyTorch
```

```
class TextClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, output_dim):
        super(TextClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.embedding.weight = nn.Parameter(embedding_matrix)  # Load
pre-trained embeddings
        self.embedding.weight.requires_grad = False  # Freeze embeddings
        self.fc = nn.Linear(embedding_dim, output_dim)

    def forward(self, x):
        embedded = self.embedding(x)
        return self.fc(embedded.mean(dim=1))  # Mean of embeddings for
simplicity

# Example usage
model = TextClassifier(vocab_size, embedding_dim, output_dim=2)
```

2. TensorFlow/Keras Implementation

```
import numpy as np
import tensorflow as tf
from gensim.models import KeyedVectors

# Load pre-trained Word2Vec or GloVe embeddings
embedding_path = 'path/to/your/word2vec_or_glove_file'  # e.g.,
'GoogleNews-vectors-negative300.bin'
word_vectors = KeyedVectors.load_word2vec_format(embedding_path,
binary=True)

# Prepare the embedding matrix
embedding_dim = word_vectors.vector_size
vocab_size = len(word_vectors.index_to_key)
embedding_matrix = np.zeros((vocab_size, embedding_dim))

# Map words to index in embedding matrix
word_to_idx = {word: idx for idx, word in
enumerate(word_vectors.index_to_key)}

# Fill embedding matrix with pre-trained embeddings
for word, idx in word_to_idx.items():
    embedding_matrix[idx] = word_vectors[word]
```

```
# Define a simple model with embedding layer using TensorFlow/Keras
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size,
output_dim=embedding_dim,
                              weights=[embedding_matrix], trainable=False),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(2, activation='softmax')  # Example output layer
for binary classification
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

Q13. How would you use cosine similarity to find the top 5 similar words to a given word using pre-trained embeddings in Python?

Answer: To find the top 5 words most similar to a given word using cosine similarity with pre-trained embeddings,

```
import numpy as np
from gensim.models import KeyedVectors

embedding_path = 'path/to/your/word2vec_or_glove_file'
word_vectors = KeyedVectors.load_word2vec_format(embedding_path,
binary=True)

# Define function to compute cosine similarity
def cosine_similarity(vec1, vec2):
    return np.dot(vec1, vec2) / (np.linalg.norm(vec1) *
np.linalg.norm(vec2))

# Find top 5 most similar words
def top_similar_words(word, word_vectors, top_n=5):
    if word not in word_vectors:
```

```
        return f"{word} not in vocabulary"

    word_vec = word_vectors[word]
    similarities = []

    for other_word in word_vectors.index_to_key:
        if other_word == word:
            continue
        sim = cosine_similarity(word_vec, word_vectors[other_word])
        similarities.append((other_word, sim))

    # Sort by similarity and get top N
    similarities.sort(key=lambda x: x[1], reverse=True)
    return similarities[:top_n]

# Example usage
top_words = top_similar_words("king", word_vectors)
print("Top similar words to 'king':", top_words)
```

Fine Tuning, GAN

Q1. Describe the mathematical concept of low-rank decomposition in LoRA. How are the matrices A and B constructed, and what constraints should be considered when choosing the rank r?

Answer:Low-rank decomposition in LoRA focuses on representing the weight update matrix ΔW as a product of two smaller matrices, A and B, to efficiently reduce the number of trainable parameters. Given the pre-trained weight matrix W_0 of size d×k, LoRA introduces an update matrix ΔW of the same size. Instead of directly updating ΔW, LoRA decomposes it into two low-rank matrices: A, of dimension d×r, and BBB, of dimension r×k, where r is the rank (typically much smaller than both d and k). Mathematically, this decomposition is expressed as:

$$\Delta W = A \times B$$

Thus, instead of training d×k parameters in ΔW, LoRA only trains r(d+k) parameters, drastically reducing the computational load, since $r \ll d,k$.

The choice of r, the rank, is pivotal. If r is too small, the low-rank approximation might not capture the necessary information for the task, leading to underfitting. If r is too large, it defeats the purpose of parameter efficiency. Hence, r needs to be chosen carefully to balance the trade-off between model expressiveness and computational efficiency.

Q2. How does block wise quantization in QLoRA with smaller blocks reduce precision loss, and how does it impact memory efficiency and model performance on resource-constrained hardware?

Answer:Blockwise quantization, as used in QLoRA, divides the weight matrices of a model into smaller blocks to apply quantization more effectively. In QLoRA, the model weights are quantized to 4-bit precision, but instead of applying this quantization uniformly across the entire matrix, the matrix is split into fixed-size blocks (typically of size 64). Each block is then independently quantized, allowing for better representation of the weights, especially in regions where the distribution of values is non-uniform. The goal is to preserve as much information as possible while reducing precision and memory usage.

This approach addresses the problem of precision loss that occurs when quantizing values that are not uniformly distributed across the range. By handling smaller blocks independently, QLoRA ensures that values within each block are represented more accurately, minimizing the error that arises from quantization. This is especially useful for large models, as it reduces the memory footprint while maintaining model performance, making it feasible to fine-tune and run inference on resource-constrained hardware like GPUs with limited memory.

Q3. Why is LoRA classified as an “adaptation” technique rather than an “adapter”? How does this classification impact the fine-tuning strategy, and why is it important not to introduce additional layers in LoRA?

Answer: LoRA is classified as an "adaptation" technique because it modifies the model's existing weight matrices by adding low-rank matrices rather than inserting new layers like "adapters." This is crucial because adding layers in adapter-based methods can lead to increased latency during inference, making the model slower in real-time applications. LoRA, on the other hand, directly updates certain parts of the model’s parameters while keeping

most of the original weights frozen. By not introducing new layers, LoRA minimizes the impact on model speed and memory usage, allowing for efficient fine-tuning. This approach is particularly beneficial for large language models, where adding layers would significantly increase both computational costs and latency.

Q4. In what scenarios might adapter-based approaches still be useful despite their higher latency compared to LoRA?

Answer: Adapter-based approaches can be useful when you need flexibility or want to handle many different tasks, such as text classification, sentiment analysis, and named entity recognition. In these cases, each task can have its own adapter, which makes it easy to switch between tasks without retraining the entire model. This is particularly helpful in multi-task learning or when you frequently update tasks, like adapting to new domains or languages.

Adapters are also useful when more changes to the model are needed than LoRA allows. For example, if you're working on complex tasks like summarization or machine translation that require significant adjustments to the model's architecture, adapters can provide that capability. Even though adapters introduce more latency, they work well in situations where speed isn't the main concern, such as offline processing or research projects, where performance on multiple tasks is more critical than real-time response.

Q5. How do perplexity and BLEU score differ in their roles as evaluation metrics for language models, and in what situations would you prioritize one metric over the other when assessing model performance?

Answer: Perplexity and BLEU score are two important metrics for evaluating language models (LLMs). Perplexity measures how well a model predicts a sequence of words. A lower perplexity indicates better predictive performance, making it useful for assessing how fluently a model can generate text. However, it doesn't directly evaluate the meaning or quality of the generated content.

In contrast, the BLEU score evaluates the quality of generated text by comparing it to reference outputs. It measures how many n-grams (sequences of words) in the generated text match those in the reference texts. This makes BLEU particularly useful for tasks like machine translation and summarization, where accuracy relative to specific outputs is essential. While perplexity focuses on the model's ability to predict words, BLEU emphasizes how closely the generated text matches expected results.

Q6. Discuss in what context it is recommended to use transfer learning and when it is not.

Answer: Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point for computer vision and natural language processing tasks given the vast computing and time resources required to develop neural network models on these problems and from the huge jumps in a skill that they provide on related problems.

Transfer learning is used for tasks where the data is too little to train a full-scale model from the beginning. In transfer learning, well-trained, well-constructed networks are used which have learned over large sets and can be used to boost the performance of a dataset.

Transfer Learning can be used in the following cases:

The downstream task has a very small amount of data available, then we can try using pre-trained model weights by switching the last layer with new layers which we will train.

In some cases, like in vision-related tasks, the initial layers have a common behavior of detecting edges, then a little more complex but still abstract features and so on which is common in all vision tasks, and hence a pre-trained model's initial layers can be used directly. The same thing holds for Language Models too, for example, a model trained in a large Hindi corpus can be transferred and used for other Indo-Aryan Languages with low resources available.

Cases when transfer Learning should not be used:

The first and most important is the "COST". So is it cost-effective or we can have a similar performance without using it.

The pre-trained model has no relation to the downstream task.

If the latency is a big constraint (Mostly in NLP) then transfer learning is not the best option. However Now with the TensorFlow lite kind of platform and Model Distillation, Latency is not a problem anymore.

Q7. What is the role of noise in GANs, and why is it necessary?

Answer: In Generative Adversarial Networks (GANs), noise plays a crucial role in generating diverse and realistic outputs. The generator takes a noise vector, usually sampled from a standard normal distribution, as its input. This randomness allows the generator to explore

different points in the latent space, enabling it to produce various data samples instead of repeating the same outputs. This is important for preventing mode collapse, where the generator creates only a limited range of outputs.

Moreover, the noise helps the generator learn to generalize from the training data, which enhances its ability to create new instances that closely match the true data distribution. In advanced GANs like StyleGAN, noise can also be adjusted to control specific features in the generated data, offering more flexibility in the synthesis process. Overall, noise is essential in the GAN framework for achieving both diversity and realism in the generated outputs.

Q8. What are the original loss functions used in GANs, and what problems are associated with them? How do Wasserstein GANs address them?

Answer: In the original Generative Adversarial Networks (GANs), the loss functions are designed for two players: the generator GGG and the discriminator DDD. The generator tries to create data that looks real, while the discriminator tries to tell apart real data from fake data. The generator's loss is calculated as:

$$L(G) = -\mathbb{E}_{z \sim p_z(z)}[\log(D(G(z)))]$$

The discriminator's loss is:

$$L(D) = -\mathbb{E}_{x \sim p_{data}(x)}[\log(D(x))] - \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

While this setup helps the generator improve, it often leads to problems like unstable training, where the generator doesn't learn effectively, and mode collapse, where it produces only a few types of outputs. To fix these issues, Wasserstein GANs (WGANs) use a different loss function based on the Wasserstein distance, which measures how different two distributions are. Instead of maximizing the log probability, the generator in WGANs tries to minimize:

$$L_{WGAN}(G) = -\mathbb{E}_{z \sim p_z(z)}[D(G(z))]$$

And the discriminator tries to maximize:

$$L_{WGAN}(D) = \mathbb{E}_{x \sim p_{data}(x)}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))]]$$

WGANs also require the discriminator to satisfy a condition called Lipschitz continuity, which keeps its output bounded. This approach helps prevent problems like vanishing gradients, leading to a more stable training process. By focusing on the distance between real and generated data rather than probabilities, WGANs provide better feedback for both the generator and discriminator, allowing them to learn more effectively.

Q9. Explain how the auxiliary classifier GAN (AC-GAN) works.

Answer: The Auxiliary Classifier GAN (AC-GAN) is an improved version of the standard GAN that adds a classifier to the discriminator. The generator in AC-GAN creates synthetic data while also receiving class labels as input. This helps it generate data that is not only realistic but also belongs to specific categories. The discriminator has two tasks: it determines whether a sample is real or fake, and it classifies the samples into their respective classes. This dual role enhances the model's ability to generate diverse and high-quality samples.

The loss functions in AC-GAN include two components for the discriminator: one for distinguishing real from fake samples and another for predicting class labels. The generator's loss function focuses on producing samples that the discriminator considers real and classifies correctly. This setup improves the training process and results in better sample generation, as it provides the generator with clearer feedback on how to create more accurate data.

Q10. Describe the truncation trick and its effect on the quality of GAN outputs.

Answer: The truncation trick is a technique used in GANs, especially in StyleGAN models, to control the trade-off between the diversity and quality of generated samples. It involves adjusting the distribution of the latent vectors (the input noise) that are fed into the generator. Normally, latent vectors are sampled from a standard normal distribution, but in the truncation trick, the latent vectors are scaled towards the mean of the distribution. This reduces their variance, effectively "truncating" the distribution.

By doing this, the generated outputs become more realistic and of higher quality because the generator is pushed to produce samples closer to the typical (mean) training data it has learned. However, the downside is that this comes at the cost of diversity—fewer unique or diverse outputs are produced since the input space is restricted. So, while the truncation trick can improve the visual quality of generated images, it may limit the model’s ability to generate a wide range of different outputs.

Q11. Discuss the importance of using multiple scales in GANs for improving output quality.

Answer: Using multiple scales in GANs, like in Progressive GANs and StyleGAN, is important for improving the quality of the generated images. The idea is to help the model learn different levels of detail, from large, basic shapes to fine, small details. This is done by starting with low-resolution images and gradually adding more layers as the model works with

higher-resolution images. This way, the model learns simpler features first and then adds more complex details, making the images look more realistic.

At lower resolutions, the generator learns overall shapes and structures, like the basic layout of objects. As the resolution increases, it learns smaller details like textures and edges. This method helps prevent the problem where the model generates similar or repetitive images (called mode collapse) and makes training more stable since the model isn't overwhelmed with too much detail at once.

The discriminator also benefits from multiple scales because it can check both the big picture and the finer details. By doing this, it gives better feedback to the generator, which helps produce images that are not only consistent in their overall structure but also sharp and detailed, leading to much higher-quality outputs.

Q12. Explain the Pix2Pix algorithm and its applications image-to-image translation.

Answer: The Pix2Pix algorithm is a type of GAN designed for image-to-image translation tasks, where the goal is to transform one type of image into another while maintaining key features. It works by using a pair of images: an input image (source) and a target image (desired output), and the model learns to map between the two. The architecture consists of a generator and a discriminator. The generator creates an output image given an input image, while the discriminator checks whether the output looks realistic compared to the true target image.

The generator in Pix2Pix is usually a U-Net, which has an encoder-decoder structure. The encoder captures high-level features, while the decoder reconstructs the output image. The U-Net uses skip connections between layers, allowing it to pass both low-level and high-level information between the input and output, which helps generate sharper images.

The discriminator in Pix2Pix uses a patch-based approach (PatchGAN), where instead of evaluating the entire image, it evaluates smaller patches of the image to check if they are realistic. This makes the model focus on local details, improving the quality of the generated images.

Applications of Pix2Pix include tasks like turning sketches into realistic images, converting black-and-white images to color, translating satellite images into maps, or even transforming day photos into night scenes. It’s widely used in domains where a paired dataset of input and output images is available, making it very versatile for various image transformation tasks.

Q13.Discuss the limitations of traditional metrics like Inception Score and FID in assessing GAN performance.

Answer: Inception Score (IS): IS evaluates GAN performance by measuring the confidence of a pre-trained classifier (Inception network) in recognizing generated images and their diversity. It focuses on how well the generated images fit into known classes. However, IS has limitations: it is biased toward the classifier's training data (e.g., ImageNet), doesn't directly compare generated images to real data, and often fails to detect mode collapse, making it less reliable for datasets outside of predefined categories.

Fréchet Inception Distance (FID): FID compares the distribution of real and generated images by using the mean and covariance of their high-level features, extracted by a pre-trained Inception network. While FID accounts for both quality and similarity to real data, it is sensitive to biases from the pre-trained model and minor image artifacts, which can skew the results. It also simplifies image features, which might miss important differences in more complex image properties like texture and structure.

Q14.How Can we Scale GANs Beyond Image Synthesis?

Answer:
Text: GANs struggle with text because it is made of discrete units (like words), and GANs work best with continuous data, which allows smooth gradients for training. To handle this, one approach is to convert text into continuous forms, letting the GAN process it without dealing directly with discrete words. Another method uses gradient estimation to work with text data directly. However, neither approach has yet matched the performance of traditional models like those based on likelihood, which remain better for tasks like language modeling.

Structured Data: For structured data, like graphs, GANs haven't been very successful. This is part of a broader challenge in geometric deep learning, where deep learning models, in general, have trouble with non-Euclidean data like graphs. Some attempts have been made, such as generating random walks from graphs, which the GAN's discriminator evaluates. Still, no significant progress has been made in applying GANs to structured data, and it's not clear whether GANs are the best choice for this type of data compared to other models.

Audio: Audio is the area where GANs have made the most progress outside of images. Audio is continuous, like images, so GANs can be adapted more easily for tasks like audio synthesis. Early models introduced special techniques to handle the continuous nature of sound. Recent advancements show that GANs can even outperform older methods, like autoregressive models, on certain audio quality measures, suggesting GANs may soon become a key tool for generating realistic audio.